KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Efficient Language Modeling for Automatic Speech Recognition

Joris Pelemans

Supervisor:
Prof. dr. ir. P. Wambacq

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

5 May 2017

Efficient Language Modeling for Automatic Speech Recognition

Joris PELEMANS

Examination committee:
Prof. dr. ir. C. Vandecasteele, chair
Prof. dr. ir. P. Wambacq, supervisor
Prof. dr. ir. H. Van hamme, co-supervisor
Prof. dr. ir. D. Van Compernolle
Prof. dr. M. Moens
Dr. ir. K. Demuynck
Prof. dr. L. Macken
(Ghent University, Belgium)
Dr. ir. C. Chelba
(Google Inc., Mountain View, USA)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

5 May 2017

Preface

This PhD thesis is about predicting what someone will say, about predicting the future. That may sound impossible, but in fact, each and everyone of us does it every day, when we anticipate the movements of other vehicles in traffic, when we look to the sky to decide whether we should take an umbrella. It is our way to cope with a tiny fraction of the infinite uncertainties that we face in our lives. By guessing what others will say or do, we can better choose our own actions, our own words. By predicting the future, we optimize our choices. But in order to make good predictions, it is important to look at the past. For it is in the past that we learn the patterns that enable us to make such predictions. I would like to take this opportunity to do exactly that: look back on this PhD and be grateful for what were without a doubt some of the best moments of my life.

I started this PhD in the fall of 2010, more than 7 years after I graduated from university. Needless to say that I was a bit rusty when it came to mathematics which meant that the start of my academic career was rather slow and teaching sessions were very stressful. However, it quickly became clear to me that both my supervisor Patrick Wambacq and my co-supervisor Hugo Van hamme were very patient and understanding, as well as helpful. While Patrick gave me the space and time to find my own research path, assisting me where needed, Hugo provided me with the necessary code and insights to successfully complete the teaching sessions. This support remained a constant over the course of the following years and it is clear that without it, I would not have been able to finish my PhD. For this, I owe both of them a great deal of gratitude.

Gratitude is also due to Kris Demuynck who suggested that I started working on the FLaVoR decoder and whose endless (sometimes too) technical explanations really taught me a lot. The work on FLaVoR eventually led to my first publication and conference in 2012 in Japan which in hindsight really was the beginning of many exciting opportunities. Even after leaving KU Leuven, Kris continued to help me with any questions I had about SPRAAK and always

gave excellent comments on my paper drafts.

It was in Japan that I met Yangyang Shi who wanted to evaluate his language models on Dutch speech. Not only did this meeting lead to a nice collaboration which resulted in a journal paper, it was also Yangyang who introduced me to the concept of word embeddings which inspired me to write a journal paper on semantic language models.

Japan was also the place where I met Ciprian Chelba. His interest in our work and perseverance to collaborate finally paid off in 2014, when I was accepted to do a 3 month internship at Google's Headquarters in California. Thank you, Ciprian, for this life-changing experience and the many opportunities that it has given me afterwards. Working with you and Noam and learning from the world's leading scientists was truly inspirational!

Being away from my family for several weeks or even months was not easy and I owe many thanks to Alan and Dan who managed to make this time a lot more enjoyable. I will never forget our trips to Yosemite and Monterey, our excellent chess parties and of course, Bryan the carpenter.

During all this time, I was surrounded by a great group of colleagues in the speech group. I have particularly happy recollections of such events as the winter barbecue, Kseniya's birthday party, StepMania at Emre's, chess with Hasan, many, many great wiki moments with Joris, Chinese new year with Xueru, Meng's candy tricks, Dino's book gift, epic rap battles with György, snorkling with Deepak, pingpong Reza style, exploring Iceland with Patrick, Yellowstone with Lyan, enjoying fancy food with Dirk, jacuzzi talk with Jeroen, kayak, paintball, climbing trees, escape room, PhD discussions, coffee breaks and many more! Thanks to all of you who made this place an awesome workplace! And a special thanks to Vincent who took the time to listen and criticize the dry-run of my preliminary defense.

A special thanks goes to Oma and Vava who were always there with kind, supporting words and who have taken great care of our children when we were not able to.

And last, but not least, I would like to thank Veerle, Elin and Isak, for giving me the ultimate motivation and at the same time the best possible perspective. Veerle, without you, I probably wouldn't have even started this PhD, let alone finished it. It can not have been easy when I was away from home, either physically, during conferences and internships, or mentally, when I had another deadline. Yet, you were always willing to give me the opportunity, to support me, for which I am eternally grateful. Elin en Isak, dank je wel om mij aan het eind van een moeilijke dag telkens weer te doen beseffen wat er echt belangrijk is in het leven.

Abstract

Since the advent of deep learning, automatic speech recognition (ASR), like many other fields, has advanced significantly. Both the acoustic model and the language model are now based on artificial neural networks which has yielded drastic improvements in recognition accuracy. However, whereas the state-of-the-art acoustic models have been integrated directly into the core of most if not all speech recognizers, the same cannot be said of language models. Current speech recognizers are still employing n-gram language models, models that were developed during the 1970s, because they achieve reasonable accuracy and, more importantly, because they are extremely efficient. More advanced language models such as the current state-of-the-art recurrent neural network language models (RNNLMs) on the other hand are computationally expensive and can therefore only be applied in a multi-pass recognition: the model is used only in a second pass to rescore the output of a first pass that uses n-grams. This is suboptimal for both speed and accuracy and indicates that there is a clear demand for alternatives.

In this dissertation, we propose five such alternatives to improve upon regular n-grams, while striving towards minimal computational complexity. First, we improve the prediction accuracy of n-gram language models without sacrificing their efficiency. To this end, we propose a class-based n-gram language model that uses compound-head clusters as classes. We argue that compounds are well represented by their head which alleviates the overgeneralization that class-based models usually suffer from. We present a clustering algorithm that is capable of detecting the head of a compound with high precision and we use aggregated statistics to model both unseen compounds as well as infrequent compounds. Our technique is validated experimentally on the Dutch *CGN* corpus and shows significant word error rate reductions compared to regular n-gram models.

In a second proposal, we overcome the inability of n-grams to capture long-distance relations between words by combining them with semantic language

models i.e. models that are able to detect semantic similarities between words. We conduct a thorough investigation of two existing semantic language models, namely cache models and models based on Latent Semantic Analysis (LSA), and compare them to a novel semantic model that is based on word embeddings. Not only does our proposed model consistently achieve higher prediction accuracy on Dutch newspaper and magazine data, it is also twice as fast as the model based on LSA and combines well with cache models.

Another approach to modeling long-distance dependencies is proposed in the form of a novel estimation technique, called Sparse Non-negative Matrix (SNM) estimation. This technique is able to incorporate arbitrary features, yet scales gracefully to large amounts of data. We show that SNM language models trained with n-gram features are a close match for the well-established Kneser-Ney models and that the addition of skip-gram features yields a model that is in the same league as the state-of-the-art RNNLM, as well as complementary. The model is validated experimentally and shows excellent results on two different data sets: Google's *One Billion Word Benchmark* and a smaller subset of the *LDC Gigaword Corpus*. Moreover, we show that a first implementation is already 10x faster than a dedicated implementation of an RNNLM.

Efficient language modeling can also be achieved by adapting the recognizer such that it can spend more resources on the language model. To this end, we propose a layered architecture, that uses the output of a first acoustic layer as input for a second word decoding layer. This decoupling alleviates the task of the decoder which makes it possible to apply more complex language models. We show on the Dutch *N-Best Benchmark* that, although we have not exploited its full potential, the architecture is already competitive to an all-in-one approach in which acoustic model, language model and lexicon are all applied simultaneously.

Finally, we propose a novel language model adaptation technique that can be applied to ASR of spoken translations. The technique consists of n-gram probability inflation using exponential weights based on translation model probabilities which reduces the number of updates. It does not enforce probability renormalization and reduces data storage and memory load by storing only the update weights. We validate this technique experimentally and show that it achieves a significant word error rate reduction on spoken Dutch translations from English, while having little to no negative effect on recognition time which allows its use in a real-time computer-aided translation environment.

Beknopte samenvatting

Sinds het intreden van *deep learning* heeft automatische spraakherkenning, net zoals vele andere onderzoeksgebieden, een grote vooruitgang gekend. Zowel het akoestische model als het taalmodel zijn nu gebaseerd op artificiële neurale netwerken wat drastische verbeteringen heeft opgeleverd in de nauwkeurigheid van spraakherkenners. Echter, hoewel deze recente verbeteringen reeds opgenomen zijn in de meeste spraakherkenners op het vlak van akoestische modellering, is dit voor taalmodellen niet helemaal het geval. De huidige spraakherkenners gebruiken immers nog steeds n-gram taalmodellen, statistische modellen die ontwikkeld zijn in de jaren '70, omdat deze modellen verrassend nauwkeurig, maar vooral ook extreem efficiënt zijn. Meer geavanceerde taalmodellen zoals modellen die gebaseerd zijn op recurrente neurale netwerken (RNN) vereisen veel rekenkracht en kunnen daarom enkel gebruikt worden om de uitvoer van een herkenner met een n-gram taalmodel te herevalueren. Dit is suboptimaal, zowel voor wat betreft de nauwkeurigheid als de snelheid.

In dit proefschrift stellen we vijf verschillende technieken voor om dit aan te pakken. In een eerste voorstel proberen we de voorspellingsnauwkeurigheid van n-gram taalmodellen te verbeteren zonder daarbij aan efficiëntie in te boeten. Om dit te bereiken stellen we een klassegebaseerd n-gram taalmodel voor dat gebruik maakt van clusters die bestaan uit samenstellingen en hun grondwoord of *hoofd*. We argumenteren dat samenstellingen goed vertegenwoordigd worden door hun hoofd en dat dit de oververalgemening beperkt die eigen is aan klassegebaseerde modellen. We presenteren een clusteralgoritme dat het hoofd van een samenstelling nauwkeurig kan detecteren en gebruiken geaggregeerde statistieken om zowel ongeziene als infrequente samenstellingen te modelleren. Experimenten op het *Corpus Gesproken Nederlands* tonen significante *word error rate* (WER) reducties in vergelijking tot standaard n-gram modellen.

In een tweede voorstel proberen we afhankelijkheden te modelleren tussen woorden die zich op grotere afstand van mekaar bevinden. We doen dit door n-

gram taalmodellen te combineren met semantische taalmodellen: modellen die semantische gelijkenis tussen woorden kunnen meten. We voeren een grondig onderzoek uit op twee bestaande semantische taalmodellen nl. cachemodellen en modellen gebaseerd op Latent Semantische Analyse (LSA), en vergelijken deze met een nieuw model dat gebaseerd is op het continue skip-grammodel, het model dat momenteel het best scoort in woordgelijkenis. Dit nieuwe model behaalt niet alleen een consistent hogere voorspellingsnauwkeurigheid op Nederlands kranten- en tijdschriftmateriaal, maar het is ook twee keer zo snel als het LSA-model en combineert goed met cachemodellen.

Een volgende poging om afhankelijkheden over lange afstanden te modelleren is *Sparse Non-negative Matrix* (SNM) schatting. Deze nieuwe schattingstechniek kan niet alleen willekeurige *features* combineren, maar is ook erg schaalbaar naar grote hoeveelheden data. We tonen aan dat SNM taalmodellen die gebruik maken van n-gram features de performantie van Kneser-Ney modellen benaderen en dat een model met n-gram en skip-gram features kan wedijveren met de allerbeste RNN modellen. De modellen worden experimenteel gevalideerd en behalen uitstekende resultaten op twee verschillende datasets: de *One Billion Word Benchmark* en een kleiner subcorpus van het *LDC Gigaword Corpus*. Tenslotte tonen we aan dat een eerste implementatie reeds 10x sneller is dan een geoptimaliseeerde implementatie van een RNN model.

Een van de obstakels die het toepassen van een complexer taalmodel in spraakherkenning bemoeilijken is het feit dat de meeste herkenners alle kennisbronnen combineren tot een grote zoekruimte. Het doorzoeken van deze enorme ruimte vereist dat elk van de kennisbronnen – het akoestisch model, het uitspraakwoordenboek en het taalmodel – eenvoudig gehouden wordt. Als alternatief stellen we een gelaagde architectuur voor die de uitvoer van een eerste akoestische laag als invoer gebruikt voor een tweede laag die instaat voor de woordherkenning. Deze ontkoppeling verlicht de taak van de herkenner wat het mogelijk maakt om in de tweede laag meer geavanceerde taalmodellen te gebruiken. We tonen aan op de Nederlandse *N-Best Benchmark* dat deze architectuur reeds kan wedijveren met de standaard architectuur, terwijl ze nog niet van haar volledige potentieel gebruik maakt.

Tenslotte stellen we een nieuwe taalmodeladaptatietechniek voor die kan gebruikt worden bij het automatisch herkennen van gesproken vertalingen. De techniek bestaat erin de bestaande n-gramkansen te verhogen door ze te vermenigvuldigen met exponentiële gewichten die gebaseerd zijn op kansen uit een vertalingsmodel zonder nadien renormalisatie toe te passen. Onze experimenten op het *Corpus Gesproken Nederlands* tonen aan dat de techniek een grote WER reductie behaalt zonder de snelheid of opslag van het systeem nadelig te beïnvloeden. Dit laat toe dat de aangepaste modellen in real time gebruikt worden in een praktische applicatie die vertalers ondersteunt.

Abbreviations

AM	Acoustic Model
ASR	Automatic Speech Recognition
CAT	Computer-Aided Translation
CELEX	Centre for Lexical Information
CGN	Corpus Gesproken Nederlands
CHC	Compound-Head Clustering
CNA	Central News Agency of Taiwan
CPU	Central Processing Unit
CSM	Continuous Skip-gram Model
EM	Expectation Maximization
FLaVoR	Flexible Large Vocabulary Recognition
FST	Finite State Transducer
G2P	Grapheme-to-Phoneme
GIS	Generalized Iterative Scaling
GMM	Gaussian Mixture Model
GT	Good-Turing
HMM	Hidden Markov Model
HSME	Hierarchical Softmax Maximum Entropy
IV	In-Vocabulary
KN	Kneser-Ney

LDC	Linguistic Data Consortium
LM	Language Model
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
LVCSR	Large Vocabulary Continuous Speech Recognition
ME	Maximum Entropy
MIDA	Mutual Information Discriminant Analysis
MT	Machine Translation
NE	Named Entity
NNLM	(Feed-forward) Neural Network Language Model
OOV	Out-Of-Vocabulary
PER	Phone Error Rate
POS	Part Of Speech
PPL	Perplexity
RNN	Recurrent Neural Network
RNNLM	Recurrent Neural Network Language Model
RNNME	Recurrent Neural Network with Maximum Entropy connections
SBO	Stupid Back-Off
SNM	Sparse Non-negative Matrix
SNMLM	Sparse Non-negative Matrix Language Model
SPRAAK	Speech Processing, Recognition and Automatic Annotation Kit
SRILM	Stanford Research Institute Language Modeling
SSTable	Sorted String Table
SVD	Singular Value Decomposition
TF-IDF	Term Frequency - Inverse Document Frequency
TM	Translation Model
VTLN	Vocal Tract Length Normalization
WB	Witten-Bell
WER	Word Error Rate
xRT	Real-Time Factor

List of Symbols

$A(f,t)$	Adjustment function with feature f and target t as input
$\mathbf{C}$	Count matrix
C_{ft}	Count of co-occurrence of feature f and target t
C_{f*}	Total occurrence count of feature f
$c(w)$	Count of word w
$\hat{c}(w)$	Count estimate of word w
$\tilde{d}_{q-1}$	Pseudo-document representing the history
E, e_1^J	Target language text
$\bar{e}$	Target language phrase
F, f_1^I	Source language text
$\bar{f}$	Source language phrase
$\mathcal{F}$	Set of features
$\mathcal{F}_+$	Set of active features
$\mathbf{H}$	Weight matrix for hidden layer of neural network
$\mathbf{h}(f,t)$	Meta-feature vector extracted from feature f and target t
$\mathbf{M}$	Feature weight matrix
O, o_1^T	Acoustic feature sequence
$P_w(\bar{f}, \bar{e})$	Lexical weight for alignment of phrases $\bar{f}$ and $\bar{e}$
q_t	HMM state at time t
$\mathbf{S}$	Matrix of singular values
$\mathcal{T}$	Training corpus
$\mathbf{U}$	Matrix of left singular vectors
$\mathbf{u_i}$	i^{th} column of matrix $\mathbf{U}$
$\mathbf{V^T}$	Matrix of right singular vectors
$\mathbf{v_j}$	j^{th} column of matrix $\mathbf{V}$
$\tilde{\mathbf{v}}_{\mathbf{q-1}}$	Projection of the history into latent space
$\mathcal{V}$	Vocabulary

W, w_1^N	Word sequence
$\mathbf{W}$	Weight matrix for output layer of neural network
$w(f\|e)$	Lexical probability for alignment of words f and e
X	Acoustic signal
$\mathbf{x^T}$	Feature activation vector
$\mathbf{y}$	Vector of target words
$\hat{\mathbf{y}}$	Vector of unnormalized (log-)probabilities
γ	AdaGrad learning rate scaling factor
Δ_0	AdaGrad initial accumulator
η	Learning rate
$\boldsymbol{\theta}$	Vector of meta-feature weights
$\phi(\bar{f}, \bar{e})$	Translation probability for phrases $\bar{f}$ and $\bar{e}$
$\Phi(w_1^{k-1})$	Equivalence classifier of the history w_1^{k-1}

Contents

List of Figures

List of Tables

Chapter 1

Introduction

When researchers started working on automatic speech recognition (ASR) some 70 years ago, it was generally believed that it would be sufficient to investigate the characteristics of the speech sound wave to extract the pronounced sounds, words and sentences. However, when at the end of the 50's, no significant progress was made, many researchers became pessimistic, believing that perhaps machines would never be able to unravel the mysteries of speech and turning their attention instead to other fields. Others were less pessimistic and concluded that "automatic speech recognition is probably possible only by a process that makes use of information about the structure of the language being recognized as well as of the characteristics of the speech sound wave" [33].

In 1957, Noam Chomsky, who is sometimes described as the father of modern linguistics, wrote his famous first book *Syntactic Structures* [20], which introduced the concept of a transformational generative grammar. In it, Chomsky argued that language was the outcome of a generative process governed by rules and that syntax and semantics were to be viewed independent of one another. The book had a great impact on the scientific communities including the speech community. This explains why the structure of language was believed to be best integrated into the recognition process by means of linguistically motivated decoders such as syntactic and semantic processors. Lindgren stated that those relatively few engineers who had become committed to the natural language automata had been obliged to "drink deep" of the linguistic mysteries [69]. Attempts at solving these mysteries came in the form of finite-state grammars which were limited to small domains such as chess playing [92] or geophysical inquiry [114]. It wasn't until Jelinek, Bahl

and Mercer published their *Linguistic Statistical Decoder* in 1975 that the first statistical language models (LMs) were proposed for automatic speech recognition [52] and until this day researchers in speech and other natural language technologies are still using the statistical paradigm to tackle the challenging task of language modeling.

This thesis is concerned with the computational efficiency of language modeling in automatic speech recognition. In this first chapter, we attempt to describe what this entails by covering its three main components: language modeling, automatic speech recognition and efficiency. We will start in Section 1.1 by explaining what a statistical language model is and how it has evolved from the relatively simple models proposed by Jelinek et al. to today's state-of-the-art neural network language models. In Section 1.2, we will explain what a traditional speech recognizer looks like and how it makes use of a language model. Section 1.3 focuses on *computational complexity*: it describes some of the issues that arise when applying a large language model to automatic speech recognition and sets the scene for the objectives of the thesis which will be the subject of Section 1.4. Section 1.5 then describes the methodology that we adopted to prove the validity of our proposed solutions. We end this chapter with an overview of the remaining chapters in Section 1.6.

1.1 Statistical Language Modeling

A *statistical language model* estimates probability values $P(W)$ for strings of words W in a vocabulary $\mathcal{V}$ whose size can be in the tens or hundreds of thousands and sometimes even millions. Typically, the string W is broken into sentences, or other segments such as utterances in automatic speech recognition, which are often assumed to be conditionally independent; we will assume that W is such a segment, or sentence.

Estimating full sentence language models [97] is computationally hard if one seeks a properly normalized probability model[1] over strings of words of finite length in $\mathcal{V}^*$. A simple and sufficient way to ensure proper normalization of the model is to decompose the sentence probability according to the chain rule and make sure that the end-of-sentence symbol `</S>` is predicted with non-zero probability in any context. With $W = w_1^N = w_1, \ldots, w_N$ we get:

$$P(w_1^N) = \prod_{k=1}^{N} P(w_k | w_1^{k-1}) \tag{1.1}$$

[1] In some practical systems the constraint on using a properly normalized language model is side-stepped at a gain in modeling power and simplicity, see e.g. [18].

where we refer to w_k as the *target word* and to w_1^{k-1} as the *context* or *history*.

Since the parameter space of $P(w_k|w_1^{k-1})$ is too large, the language model is forced to put the context w_1^{k-1} into an *equivalence class* determined by a function $\Phi(w_1^{k-1})$. As a result,

$$P(w_1^N) \cong \prod_{k=1}^{N} P(w_k|\Phi(w_1^{k-1})) \tag{1.2}$$

Research in language modeling consists of finding appropriate equivalence classifiers Φ and methods to estimate $P(w_k|\Phi(w_1^{k-1}))$. Once the form $\Phi(w_1^{k-1})$ is specified, only the problem of estimating $P(w_k|\Phi(w_1^{k-1}))$ from training data remains.

In the following sections, we will describe some of the most popular language models, starting with the basic n-grams and ending with the current state-of-the-art recurrent neural network language models.

1.1.1 N-grams

Arguably the most successful paradigm in the history of language modeling uses the *n-gram* equivalence classification, that is, defines

$$\Phi_{\text{n-gram}}(w_1^{k-1}) \doteq w_{k-n+1}, w_{k-n+2}, \ldots, w_{k-1}$$

with n typically between 2 and 5. Thought at the time of their introduction to be unlinguistic, most of all by Noam Chomsky, n-grams perform surprisingly well despite their obviously crude assumption that the entire context w_1^{k-1} can be reduced to a sequence of only $n-1$ words w_{k-n+1}^{k-1}.

N-gram probabilities can be estimated by means of relative frequencies:

$$P(w_k|w_{k-n+1}^{k-1}) = \frac{C(w_{k-n+1}^{k})}{C(w_{k-n+1}^{k-1})} \tag{1.3}$$

where $C(\cdot)$ indicates the count or number of times a word sequence occurs in the training data. The problem with this maximum likelihood estimate however, is that it does not take into account *data sparsity*. That is, even in a large quantity of training data, not every possible word sequence will be observed, leading to zero counts, hence zero and undefined (zero divided by zero) probabilities. Since Eq. (1.1) now factorizes the probability of a word sequence $P(w_1^N)$ into a product of conditional probabilities $P(w_k|w_{k-n+1}^{k-1})$, this

means that a single zero or undefined conditional probability $P(w_k|w_{k-n+1}^{k-1})$ leads to a zero or undefined probability $P(w_1^N)$ of the entire word sequence.

For decades researchers tried to overcome these sparsity issues with well-known smoothing techniques such as discounting [42], back-off [56] and interpolation [53]. We will not dwell on this multitude of smoothing techniques, but rather refer the interested reader to Chen and Goodman [17], who made an excellent overview in addition to proposing *modified Kneser-Ney smoothing*, which is commonly accepted as the best n-gram smoothing technique.

Even today, some 40 years after they were introduced, n-grams continue to be a popular, if not the most popular, choice of language models in automatic speech recognition and other fields. This success can be mostly explained by the ease with which they can be trained and evaluated and by the fact that they can be readily integrated into speech recognition systems. We will come back to this in Section 1.3 where we discuss the efficiency of language models and automatic speech recognition in general.

Although n-gram features reduce the parameter space of $P(w_k|w_1^{k-1})$ and consequently to some extent also the data sparsity, they completely fail to capture language phenomena that span longer distances. Moreover, even with reduced contexts, data sparsity is still an issue. For an n-gram language model to make a reliable probability estimate of a word sequence, this sequence has to occur exactly in the training data. That is, n-grams cannot generalize to similar words, which hampers their ability to share contexts and make full use of the available training data.

In the following sections, we will discuss more advanced language models that partly overcome these issues, either by explicitly using long-distance features or by conditioning the word probabilities on a more compact representation of the context.

1.1.2 Maximum Entropy

The most frequently used model to mix arbitrary features is *Maximum Entropy* (ME) [96]. ME is a member of the family of exponential models which means that $P(w_k|\Phi(w_1^{k-1}))$ takes the following log-linear form:

$$P(w_k|\Phi(w_1^{k-1})) = \frac{\exp(\hat{y}_{w_k})}{\sum_{t' \in \mathcal{V}} \exp(\hat{y}_{t'})} \tag{1.4}$$

where $\hat{\mathbf{y}}$ is the vector of unnormalized log-probabilities and each $\hat{y}_{t'}$ represents the unnormalized log-probability for a potential target word t'. How $\hat{\mathbf{y}}$ is

computed, depends on the model in question. For a ME model with features $\mathcal{F}$, $\hat{\mathbf{y}}$ can be represented as follows:

$$\hat{\mathbf{y}} = \mathbf{x}^{\mathbf{T}}\mathbf{M} \tag{1.5}$$

where $\mathbf{x}$ is a feature vector of size $|\mathcal{F}|$ and $\mathbf{M}$ is a $|\mathcal{F}| \times |\mathcal{V}|$ feature weight matrix. The features are typically binary, although continuous features have also been explored [116]. The weights can be estimated by the *Generalized Iterative Scaling* (GIS) algorithm [96], but recently it was shown that more convenient gradient-based techniques can also be used [70].

ME is interesting because it is capable of mixing different types of features, whether they are extracted from short or from large context windows. There exist many features that are extracted from larger context windows and that are therefore better suited to exploit long-span phenomena than n-grams. Below we will give a brief overview of the most popular such features without the intention of being exhaustive.

Skip-grams

Skip-grams [50, 78, 96] are features that attempt to overcome the n-gram sparsity issue by allowing words to be skipped. They are a popular choice to deal with long distances and are in fact the main features in the model that we propose in Chapter 4, where we will discuss them in more depth.

Bags of words

Another way to leverage longer distances is by ignoring the order in which the context words appear. The resulting *bag-of-word* features are very common in the field of information retrieval where syntax is often of secondary importance compared to semantics. Indeed, to some extent word order is a syntactic phenomenon, whereas semantics is more concerned with word meaning. Bag-of-word features can be used as is, but often undergo one or more transformations in an attempt to filter out or at least suppress the effect of non-informative words such as function words or irrelevant content words.

Because they contain no word order information, language models based on bag-of-word features are not suited as stand-alone models, but should always be combined with another model that does have word order information. Examples of such combinations can be found in [5], where latent semantic information (at document level) is combined with n-grams and in [108], where both syntactic and topic-based modeling are integrated with n-grams in a unified approach.

We will discuss this type of feature in more detail in Chapter 3, where we propose a new bag-of-words language model.

Syntactic Features

Syntactic features can be regarded as a special form of skip-grams, in which words are skipped if they are not part of a particular syntactic construction. Although it seems logical to include features that contain syntactic information – human language is governed by syntax after all – in practice these features are almost never used in language modeling. This is mainly due to the fact that syntactic parsing is a largely unsolved problem in itself: there are a lot of attachment ambiguities in sentences that cause the hypothesis space to grow exponentially large. Nevertheless, there have been some efforts to apply such features to language modeling, most notably [13].

1.1.3 Feed-forward Neural Networks

Although ME is capable of leveraging long-distance context, with binary features it does not have the capacity to find relationships between words that enable generalization. Even with continuous features, the model still requires feature engineering i.e. one has to decide in advance which features are likely to be informative. Instead, it is also possible to have the model learn the features automatically. Figure 1.1 shows a *Feed-forward Neural Network language model* (NNLM) [7] which is another type of log-linear model that instead of a linear combination of engineered features uses the output of a feed-forward neural network.

The unnormalized log-probabilities $\hat{\mathbf{y}}$ of a feed-forward neural network are computed as follows:

$$\hat{\mathbf{y}} = g(\mathbf{x}^{\mathbf{T}}\mathbf{H})\mathbf{W} \tag{1.6}$$

where $g(\cdot)$ is the activation function of the hidden layer (typically a tanh or sigmoid) and $\mathbf{W}$ and $\mathbf{H}$ are weight matrices for the output and hidden layer respectively. $\mathbf{x}$ is again a feature vector, but contrary to ME it does not consist of engineered features, but is instead a concatenation of learned continuous feature vectors. That is, each input feature is first projected onto a low-dimensional continuous space using a linear layer with weights $\mathbf{H}'$ that are shared over all the input features. When the input consists of n-grams, the projection of each of the $n-1$ input words – also called a *word embedding* – allows the network to discover fine-grained similarities between words. Although other features have been investigated [40], the features are typically limited to n-grams because of the complexity of the model. This is especially true when the vocabulary is large [100] which means that this model is not so interesting to model long-span dependencies.

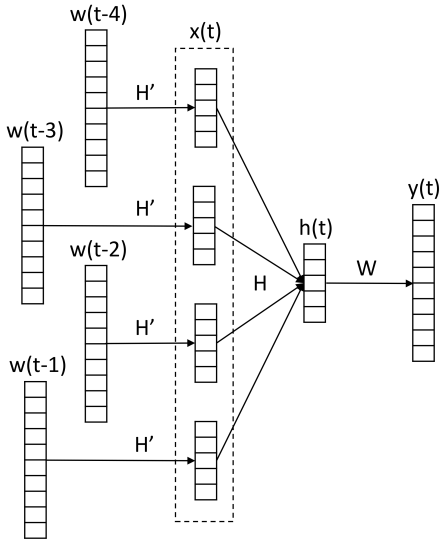

Figure 1.1: Feed-forward Neural Network Language Model with 5-gram features projected onto a low-dimensional continuous space.

1.1.4 Recurrent Neural Networks

Recently, *recurrent neural networks* (RNNs) [70] have shown excellent performance in language modeling [14, 112]. RNN language models, shown in Figure 1.2, are based on a specific type of recurrent neural network called *Elman network* [39] where the network has a recurrent connection from the hidden layer to the input layer. That is, a copy of the hidden state at time step $t-1$ is made which is used together with the current word as input for the hidden state at time step t. The current word can be encoded as a one-hot vector or, similar to the NNLM, as a projected continuous vector. The unnormalized log-probabilities $\hat{\mathbf{y}}$ are computed as in Eq. (1.6) with the difference that $\mathbf{x}$ now represents a concatenation of the vector representing the current word and the vector representing the previous hidden state.

The recurrent connection essentially gives the network a short-term memory which in theory enables the model to memorize the entire history. In practice however, the capacity of its memory is limited in time, as it turns out that RNNLMs suffer from the so-called *vanishing and exploding gradients* [47], by-products of backpropagation which is the most popular method to train a neural network. One solution to this is the *Long Short-Term Memory* (LSTM) [48]. An LSTM is a recurrent neural network model that uses the biologically inspired

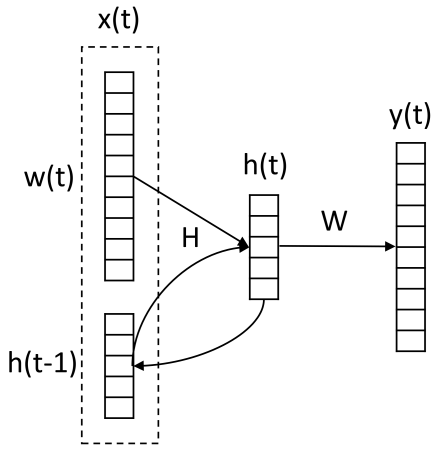

Figure 1.2: Recurrent Neural Network Language Model with one-hot encoding of the input word.

concept of gated neurons: each neuron has three different gates that control how its internal memory is used over time. The forget and input gate control how the current memory is adapted by deciding how much of the memory is kept and how much it is updated, respectively. After the memory is adapted, the output gate then decides how much of the memory is used to predict the next word in the sequence. This gating mechanism enables the model to exploit larger histories which in turn leads to higher predictive power and has made LSTMs the state of the art in language modeling [55]. However, being another member of the family of log-linear models, this model is again hampered by its computational complexity. We will come back to log-linear models and their computational complexity in Section 1.3.

1.2 Automatic Speech Recognition

The birth of statistical language models was also the birth of statistical speech recognition. Its father, Frederick Jelinek, had a background in information theory and treated automatic speech recognition as a *noisy channel problem*. The idea of the noisy channel model for speech is that an acoustic waveform is a distorted version of an intended word sequence i.e. a version that was communicated via a noisy channel. If one could build a model for the noisy channel, then in theory one could discover the original word sequence by

presenting all the existing word sequences to the noisy channel and compare the output with the given acoustic waveform. The output of the word sequence that best matches the waveform is then selected as the solution of the problem.

In the noisy channel paradigm, the goal of a speech recognizer is to find the most probable word sequence $\hat{W} = w_1, \ldots, w_N$ for a given acoustic signal X. Using Bayes' rule we can write this as follows:

$$\begin{aligned} \hat{W} &= \arg\max_W P(W|X) \\ &= \arg\max_W \frac{P(X|W)P(W)}{P(X)} \\ &= \arg\max_W P(X|W)P(W) \end{aligned} \tag{1.7}$$

where $P(X)$ can be omitted from the denominator because it is independent of W.

Eq. (1.7) indicates that the best matching word sequence can be found by maximizing the *posterior probability* $P(W|X)$, or equivalently by maximizing the product of two factors: the *prior probability* $P(W)$ and the *observation likelihood* $P(X|W)$. Estimating these two factors is the task of the main components in a speech recognition system.

$P(W)$ is the prior probability of the word sequence W and is computed by the *language model*. It expresses the well-formedness of W before observing the acoustic signal i.e. how likely is this sequence of words to be observed in a given target language. Language models are the main subject of research of this thesis and have been covered in Section 1.1.

$P(X|W)$ is the conditional probability of the acoustic signal X given the word sequence W and is computed by the *acoustic model*. Although attempts have been made [98], the acoustic model typically does not operate directly on the waveform, but instead takes its input from a *feature extraction module* or *front-end*. This module chops the waveform into pieces and extracts feature vectors O from each piece in an attempt to optimize the representation of its contents. The acoustic model then uses these features to calculate the likelihood $P(O|W)$.

Analogous to the waveform, any hypothesized word sequence W is also chopped into pieces, words, which are considered to be realized acoustically by a sequence of acoustic units, often *phonemes* (see Section 1.2.3). The pronunciation of each word is stored in a *pronunciation lexicon*. The task of the acoustic model is then to compare these acoustic units to the feature vectors extracted from the waveform. Two methods exist for such a comparison. In *template-based matching*, one builds a database of feature vectors for each

acoustic unit and matches the given feature vectors to the ones in the database. In *statistical models*, one builds a statistical model for each acoustic unit. In this thesis, we will not be using template-based matching and will therefore not discuss it in further detail, but instead refer the reader to [24].

To sum it all up: the front-end extracts feature vectors from the acoustic signal. The acoustic model uses these features to match the frames to phonemes. The lexicon maps sequences of phonemes onto words and finally, the language model composes these words to build word sequences. In the rest of this section, we will address each of these modules in a bit more detail and explain how they are all combined in the *decoder* i.e. the search engine of the speech recognizer.

1.2.1 Front-end

Although many different feature extraction schemes exist, most of the ASR systems start from *Mel spectral features*. These features adequately describe the spectral contents of the speech signal, but because of their high dimensionality and large correlation, traditional speech recognizers do not use them as is. Instead, they perform a second series of preprocessing steps that consists of dimensionality reduction and/or decorrelation. However, as this thesis is not involved with feature representation and the more recent acoustic models based on deep neural networks no longer perform these additional steps, we will limit ourselves to the description of Mel spectral features and refer the reader to the numerous articles published on this topic [2, 23, 46].

The first step in the creation of Mel spectral features, shown schematically in Figure 1.3 is applying a *pre-emphasis* filter that boosts the amount of energy in the high frequencies. This is because on average, human speech is attenuated by about 6dB per octave. Boosting the high frequency energy allows us to exploit information from the higher frequency ranges as well as the lower frequencies.

Once the energy in the high frequencies is boosted we are ready to extract features from the acoustic signal. Since we would like these features to correspond to single sounds, we will have to slice the acoustic signal into smaller pieces. Although speech is a time-variant signal, the physical limitations of the muscles in the human vocal tract allow us to assume that over a short window of time the vocal tract is constant. Moving this window over the acoustic signal, we end up with consecutive frames from which we extract the features. This windowing process is defined by three parameters: the shape of the window, the size of the window and the frame shift i.e. the time by which the window is shifted. In a typical system we slide a 30ms Hamming window over the signal in time steps of 10ms, thereby obtaining overlapping frames.

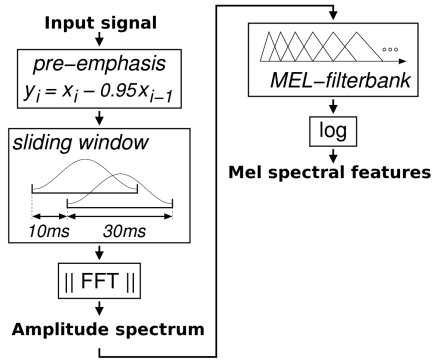

Figure 1.3: Schematic overview of Mel spectral feature extraction.

A *Discrete Fourier Transform* is computed for each of the overlapping frames, which transforms the signal from the time domain to the frequency domain. This transformation is motivated by the human ear and analyzes to what extent the different frequencies are present in the frame. Formant frequencies – frequencies with high energy – correspond to the resonance frequencies of the vocal tract and therefore give a lot of information as to which sound was actually produced. The human ear however is not equally sensitive to each frequency. For this reason, a set of overlapping band-pass filters, called a *Mel filter bank*, is applied that assigns more parameters to more interesting regions in the frequency domain. Finally the output of this filter bank is mapped to a logarithmic scale which is in accordance with human perception of loudness.

1.2.2 Acoustic Model

The features extracted in the front-end act as input for the acoustic model whose task it is to judge how probable a certain phoneme is, given the observed features. Recall from Eq. (1.7) that this corresponds to maximizing the acoustic likelihood $P(X|W)$ or equivalently $P(O|W)$ as the front-end converted the acoustic signal X into a sequence of feature vectors O. In a traditional speech recognizer, this task is usually taken care of by a *Hidden Markov Model* (HMM), which is a statistical model that assumes data is generated by a sequence of states. At each discrete point t in time an acoustic signal resides in a single state q_t of the model. From each point in time to the next, the signal makes a state transition, the probability of which is governed by the model and stored in a transition matrix. An HMM is a type of Markov Model which means that it assumes the Markov property i.e. each state q_t is only dependent on the

previous state q_{t-1}:

$$P(q_t|q_1, \ldots, q_{t-1}) \doteq P(q_t|q_{t-1}) \tag{1.8}$$

It is hidden because its states cannot be directly observed. Observations made at each point in time are considered emissions of the hidden state and are governed by a probability distribution. Until recently, the most popular method for computing emission probabilities was the *Gaussian Mixture Model* (GMM), but nowadays neural networks have taken over the field.

In order to train an acoustic model, first a large HMM is constructed as a sequence of phoneme HMMs, each of which typically contains three states. The next step is to estimate the model parameters i.e. the state transition probabilities and the parameters that define the emission probability distribution. This is usually done by the *Baum-Welch* algorithm [4] which is a type of *Expectation Maximization* (EM) algorithm [27]. For further details about HMM training we refer the reader to other literature such as [49].

1.2.3 Pronunciation Lexicon

The *pronunciation lexicon* is the interface between the feature vectors of the acoustic model on the one hand and the words of the language model on the other hand. It follows the *beads-on-a-string* paradigm which states that words can be written as sequences of acoustic units called *phonemes* i.e. the smallest meaningful units in speech. A phoneme is an abstract, language-dependent concept that comprises different realizations of sounds, called *phones*. For example, although Dutch has several different realizations of the phoneme 'r', using one or the other does not change the meaning of a word containing this phoneme.

By mapping words onto phonemes, the pronunciation lexicon allows the acoustic model to compare the observed signal with a hypothesized word sequence. Such a mapping is not straightforward as most, if not all words, can be pronounced in many ways. Not only do pronunciations differ from region to region e.g. American vs. British English *tomato*, they also differ from speaker to speaker in the same region. Even the same speaker may pronounce a word differently, depending on the context. In read speech for example one will pronounce the different phonemes explicitly, whereas in conversational speech some phonemes might be less pronounced or even omitted. To this end a lexicon can provide multiple different pronunciations for words. If the necessary data is available, the lexicon might also provide estimated probabilities for each pronunciation to indicate that some pronunciations are more likely than others.

1.2.4 Language Model

We have already mentioned that the task of the language model is to compute the prior probability $P(W)$ of the word sequence W before observing the acoustic signal X. This probability will narrow down the huge set of acoustically plausible hypotheses to those that are also plausible from a language point of view. This is especially important when the acoustic model has difficulties classifying sounds e.g. in noisy environments. However, even if the acoustic model would perfectly classify sounds to phonemes, it would still benefit from the language model as there are multiple ways to segment the string of phonemes into words by choosing different word boundaries. Moreover, the acoustic realization of a word does not always match the canonical pronunciation found in the lexicon. The language model can promote an acoustically improbable hypothesis to overcome this mismatch.

In Section 1.1, we showed how language model probabilities can be estimated from a large corpus of training data. We did not however discuss the nature of this data. For automatic speech recognition, this data is preferably spoken language, because it is in many ways different from written language. Spoken language often shows a lot of disfluencies which are not found in written language e.g. vocalized pauses, repetitions, restarts, self-corrections, ... Its word use is also different from written language with speakers often preferring shorter words and taking a more subjective viewpoint. However, since transcribing spoken data is costly, such data is limited, and in practice, language models are therefore mostly trained on huge text corpora, potentially accompanied by a smaller set of transcriptions. The language model can be trained on all of the data at once, or alternatively, several models can be built and combined in such a way that the more relevant models get a higher weight in the combination. In this way, even though the amount of transcribed speech is relatively small, the transcriptions can still contribute quite a lot to the combined model.

1.2.5 Decoding

It is the task of the decoder to find the best matching word sequence according to Eq. (1.7), using all of the above described knowledge sources. A schematic overview of this task is shown in Figure 1.4. Since words are considered to be sequences of phonemes and phonemes are sequences of HMM states, Eq. (1.7) can be equivalently written as follows:

$$\hat{w}_1^N = \arg\max_{w_1^N} P(w_1^N) \sum_{q_1^T} P(o_1^T, q_1^T | w_1^N) \tag{1.9}$$

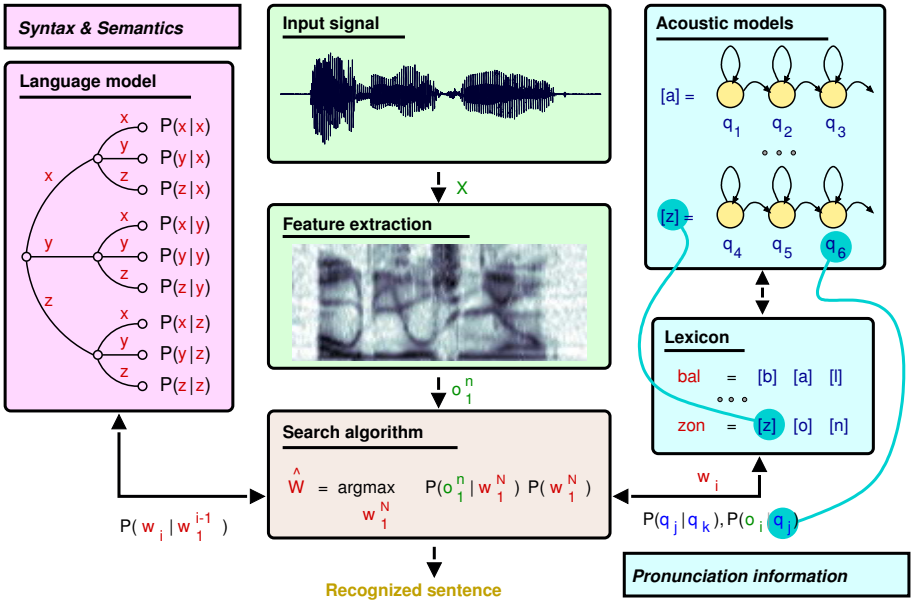

Figure 1.4: Schematic overview of a traditional speech decoder.

where q_1^T stands for any HMM state sequence of length T being emitted by the word sequence w_1^N and o_1^T represents the corresponding observations. What makes the decoding problem challenging is the combinatorial nature of possible state sequences and the fact that little or no hard decisions can be made at the low level: both the phones and the boundaries between them are uncertain. Therefore, the recognizer has to search through all possible word sequences including all possible start and end times for each word and for each of the phones of that word, taking into account all its different pronunciations. This results in a huge search space of hypotheses that have to be explored in order to find the best possible word sequence $\hat{W}$. As with all search problems there are two ways to go about this problem: breadth-first and depth-first, which in the case of speech recognition correspond to *time-synchronous* and *time-asynchronous* search, respectively.

Time-synchronous Search

In time-synchronous search all of the hypotheses at the same point in time are explored in parallel. In practice this is usually done by the *Viterbi* algorithm [111]. Viterbi is a kind of *dynamic programming* [6] that exploits the structure of the search graph: the knowledge sources used in ASR are naturally structured in a four-level hierarchy with successively, the acoustic

HMM states, the phonemes, the words and finally the sequences of words up to the sentence level. If we restrict the language model to the set of finite-state language models (n-grams and others), it can be shown that the combination of the knowledge sources yields a search graph that complies with the first-order Markov assumption. Using the fact that possible continuation paths through the graph only depend on the current position in the graph and not on how one arrived in that state, the Viterbi algorithm is able to efficiently find the best path through the search graph in $O(Q^2T)$ time with Q the number of states in the search graph. To further reduce the amount of computations, Viterbi also replaces the sum in Eq. (1.9) by a maximum:

$$\hat{w}_1^N \approx \arg\max_{w_1^N} P(w_1^N) \max_{q_1^T} P(o_1^T, q_1^T | w_1^N) \tag{1.10}$$

Even though the Viterbi algorithm is an efficient way to explore the search graph, it is still impractical to investigate every word at every point in time, especially when the vocabulary of the speech recognizer is large. Instead, low-probability paths are pruned at each time step and are not extended to the next state. A popular choice of pruning algorithm is *beam search* [106] where the number of hypotheses at each time step is not allowed to be larger than a specified beam. This beam can be defined either by setting an absolute maximum amount of hypotheses, by setting a threshold to how much the score of a hypothesis can drop below the score of the most likely hypothesis, or both.

To further improve the efficiency of the decoder, the search graph can be created in advance. A very common method that is used in e.g. the Kaldi speech recognizer [90] is to represent the search graph as one large *Finite State Transducer* (FST). FSTs are a type of finite state automata that map between two sets of symbols. Each state transition is labeled with an input symbol and an output symbol, possibly accompanied by a weight. Therefore, traversing a weighted FST corresponds to mapping from an input symbol sequence to an output symbol sequence, thereby accumulating the corresponding weights or costs. FSTs can be efficiently inverted, minimized and most importantly composed which make them an ideal representation for the knowledge sources in a speech recognizer [76]. The downside of this *static search graph* is that in order to fit in memory, restrictions on the size of the vocabulary and language model need to be imposed. For this reason, other speech recognizers such as SPRAAK [28] use a *dynamic search graph* in which the language model is combined with the other resources on the fly.

Although beam search is still the most popular search algorithm, it is not without disadvantages. First of all, this decoding strategy is not admissible which means that it is not guaranteed to find the optimal solution. Since the beam has to be limited, there is a risk that the correct solution is pruned.

Viterbi also assumes that the first-order Markov property is never violated. This means that it cannot take complete advantage of any language model that cannot be converted to an FST. Moreover, processing the different hypotheses in parallel becomes too expensive when a large number of word hypotheses need to be kept for a long time e.g. due to long n-gram contexts. The fact that it is even possible to use n-gram language models with $n > 2$ in a speech decoder is largely due to the fact that these models very often back off to a lower order due to sparsity which prevents a combinatorial explosion. This is a serious restriction as the state-of-the-art language models and also some of the proposed models in this thesis are conditioned on long histories, spanning the entire sentence or even crossing sentence boundaries.

Time-asynchronous Search

One alternative to the time-synchronous beam search described in the previous section is to use a time-asynchronous search algorithm. For speech decoding this is typically done by *stack decoding* which is a variant of the heuristic A* search, a best-first search algorithm. Rather than extending all the hypotheses in parallel, stack decoding only extends the most promising hypothesis and compares the extensions to the hypotheses from previous time steps which are stored in a priority queue or stack. The advantage of this type of decoding is that the memory usage is limited and that the search algorithm always has complete access to the history which is a very attractive property for long-span language modeling. This also means that a stack decoder is guaranteed to find the best path, at least in theory.

The reason why stack decoding is not the most popular however, is that it can take a long time before the best path is found. The efficiency of this algorithm depends on two things: selecting the likely next words and computing a score for the new hypotheses. A popular choice for the selection of likely next words is called *fast matching* and is often based on the concept of a lexical prefix tree which stores the pronunciations of words in such a way that the computation of the acoustic likelihood is shared for words with the same prefix. This allows the decoder to efficiently reduce the amount of hypotheses that need to be extended.

Once a shortlist of likely words has been made, the new hypotheses need to be scored and compared to the hypotheses on the stack. In contrast to Viterbi decoding this comparison is not trivial. Viterbi decoding is time-synchronous which means that each of the hypotheses have seen the same stretch of input signal. Time-asynchronous algorithms on the other hand compare hypotheses of different length and it is therefore not possible to use the simple product of acoustic likelihood and prior language model probability given by Eq. (1.7). Hypotheses of longer length will always have a lower posterior probability

because this probability is the result of a product of several word language model probabilities.

Instead, a family of scores $f^*(p)$ is used that were first proposed in the A* algorithm:

$$f^*(p) = g(p) + h^*(p) \tag{1.11}$$

where the estimated score $f^*(p)$ of path p depends on the score $g(p)$ of the path until now and a heuristic estimate $h^*(p)$ of the best continuation of this path to the end of the utterance.

Stack decoding is an interesting alternative to Viterbi decoding and it gives the hope of applying long-span language models to find the correct solution. However, it stands or falls with its heuristic function and until now no heuristic function has been found that efficiently computes the best path through the search space.

1.3 Efficiency

Selecting which components to integrate into a speech recognizer is a balance exercise between efficiency and modeling power, speed and quality. With increased speed typically comes decreased quality and vice versa. Although we want the transcriptions of the acoustic signal to be as accurate as possible, it would be impractical if this comes at the cost of high latency. In this section, we show that when it comes to the selection of the language model, the choices are rather extreme. That is, it is hard to find a middle ground between efficiency and modeling power.

1.3.1 N-grams: Fast, but Weak

Since most practical systems require low latency, they typically opt for n-gram language models. Although n-gram models are very simplistic, they are also quite robust with regards to spontaneous speech effects such as hesitation, lapsus, ... Because of this, they perform surprisingly well and what's more important, they are also extremely efficient. An n-gram language model is essentially a look-up table: no probability needs to be computed at test time, except when the model backs off and then the computation is easy. Moreover, n-grams can be converted to FSTs which means that they can be easily integrated into the efficient FST-based search engine which uses Viterbi decoding.

However, when we target large models with lots of training data and large vocabularies, it has been shown that n-grams are not the best choice with regards to modeling accuracy. Williams et al. [112] showed that n-gram performance quickly saturates with increased n and that an n-gram model is unable to exploit an increase in data beyond a certain point. Essentially this means that n-gram models can only take us so far. If we want to increase the accuracy of our speech recognizer we have to look for other models.

1.3.2 Log-linear Models: Powerful, but Slow

In Section 1.1, we presented several advanced language models, all of which were members of the class of log-linear models. Maximum Entropy and neural network-based language models are by far the most popular language models when one targets accuracy rather than efficiency. They perform a lot better than n-gram models, but unfortunately this comes at a cost. The problem with log-linear models lies in the denominator of Eq. (1.4):

$$P(w_k|\Phi(w_1^{k-1})) = \frac{\exp(\hat{y}_{w_k})}{\sum_{t' \in \mathcal{V}} \exp(\hat{y}_{t'})}$$

The computation of this denominator requires summing over all of the words in the vocabulary. For large vocabularies this computation is extremely expensive, which means that both training and evaluating these models becomes computationally complex. Moreover, both ME and RNNLMs are long-span language models, so the use of Viterbi decoding would be non-trivial if at all possible. It is worth noting however, that even though a more advanced language model is typically more complex, its increased predictive power should in theory help restrict the search space to viable hypotheses only which means that less hypotheses need to be evaluated.

1.3.3 Multi-pass Recognition

One possible solution to adopt more powerful language models is to use a so-called multi-pass decoding strategy. In this strategy a first pass over the data uses a simple model, typically an n-gram model, and rather than outputting a single best hypothesis, the recognizer outputs multiple hypotheses. In a second pass this reduced number of hypotheses can then be rescored with a more advanced model. The output of a recognizer can be stored as an *N-best list* or as a *lattice*. N-best lists are simply lists of the N hypotheses with the highest scores according to the first pass. They are very simple to rescore, but they are

an inefficient representation of the possible hypotheses. For example, if we want to store only 2 options for each word in a sentence with 10 words, an N-best list requires a length of 2^{10} to store all possible sentences. For this reason it is not uncommon to see N-best lists with several thousand hypotheses for each sentence. A lattice on the other hand is a more concise graph representation of the recognizer's output which allows for many more hypotheses at the same storage cost. In addition to the words that form the hypotheses and their acoustic scores, it also stores the begin and end times of each of these words.

The downside of a multi-pass approach is that by the time the output is to be rescored by the more advanced model, the optimal hypothesis given the advanced model might already be pruned out. Nevertheless, to be able to use more advanced language models this is currently the only possibility. There is a clear demand for other solutions which brings us to the objectives of this thesis.

1.4 Objectives

In this thesis, we want to propose solutions that enable the use of more powerful language models in automatic speech recognition without having to resort to multi-pass decoding. We do this in three different ways.

First, we focus our attention on the language model itself. Our goal here is to find new models that are more powerful than simple word-level n-grams, yet do not suffer from the computational complexity of log-linear models. We do this by focusing on the model's ability to generalize over related words and/or to extend the context past the n-gram window. If possible, we want to apply the models directly into a speech recognizer. If not, we want to ascertain ourselves that the models exhibit some qualities that make them more likely candidates than log-linear models and provide suggestions as to how they could be integrated into a speech recognizer.

Second, we focus on the decoding part. Our goal here is to find a new speech recognition architecture that makes it possible to decode speech using more powerful models than n-grams without having to prune valuable word hypotheses in a first pass.

Finally, we focus on language model adaptation. Language is hugely influenced by the context within which it is being used. This context can be geographical, historical, sociological, emotional, ... As a result of this inherent variability, the lexical, syntactic, or semantic characteristics of the discourse in the training and recognition tasks are likely to differ. For this reason language models can

be made more powerful by adapting them to this context. Our goal here is to find an efficient adaptation scheme that enables the language model to adapt to the context without introducing a significant overhead.

1.5 Methodology

In the previous section, we described three different objectives, all of which aim to improve both the quality and efficiency of language modeling for ASR. We will pursue these objectives by proposing new models and techniques that we will validate by means of experiments. If we want the outcome of these experiments to give us some confidence of whether we have achieved our goals, we need an objective measure to quantify our improvements. In this section, we describe two measures of quality and one measure of efficiency that we will adopt throughout the thesis.

1.5.1 Perplexity

A statistical language model can be evaluated by how well it predicts a string of symbols $W_t = w_1, \ldots, w_N$ – commonly referred to as *test data* – generated by the source to be modeled. A commonly used quality measure for a given model M with probability distribution P_M is related to the entropy of the underlying source and was introduced under the name of *perplexity* (PPL) [54]:

$$PPL_M(W_t) = \exp\left(-\frac{1}{N} \sum_{k=1}^{N} \log P_M(w_k | w_1^{k-1}) \right) \tag{1.12}$$

To give intuitive meaning to perplexity, it represents the number of guesses the model needs to make in order to ascertain the identity of the next word, when running over the test word string from left to right. It can be easily shown that the perplexity of a language model that uses the uniform probability distribution over words in the vocabulary equals the size of the vocabulary; a good language model should of course have lower perplexity, and thus the vocabulary size is an upper bound on the perplexity of any sensible language model.

Very likely, not all words in the test string W_t are part of the language model vocabulary. It is common practice to map all words that are *out-of-vocabulary* (OOV) to a distinguished *unknown word* symbol, and report the OOV rate on the test data – the rate at which one encounters OOV words in the test string

W_t – as yet another language model performance metric besides perplexity. In the usual case of *open vocabulary* language models the unknown word is assumed to be part of the language model vocabulary and its occurrences are counted in the language model perplexity calculation. A situation less common in practice is that of *closed vocabulary* language models where all words in the test data are part of the vocabulary.

1.5.2 Word Error Rate

While perplexity is a common and useful measure of quality which can be computed very quickly, it has its disadvantages. Firstly, it only assigns a value to the words that are actually observed, and completely ignores the rest of the distribution. It does not provide any insight into whether and which hypotheses are competing. Secondly, it is not an extrinsic, but an intrinsic measure i.e. it measures the quality of the model without taking into account the actual task in which the model will be employed. Reducing model perplexity does not always correspond to improved task accuracy, which has been shown on many occasions in automatic speech recognition, one of the reasons being precisely the fact that the rest of the distribution is not taken into account.

The performance of a speech recognizer is typically measured by the word error rate (WER). This measure considers three types of recognition errors: insertions, deletions and substitutions. The error rate is then computed as follows:

$$\text{WER} = \frac{C_{ins} + C_{del} + C_{sub}}{N} \tag{1.13}$$

where C denotes the amount of errors for each type and N denotes the total number of words in the reference transcription.

1.5.3 Measuring Efficiency

The above-mentioned measures are common measures of language model quality. This thesis however, is mostly concerned with language model efficiency. It is therefore important that we also quantify any progress that we make with regards to efficiency. In this thesis we chose to report runtimes. Runtimes for training the model, for evaluating it or for executing the different tasks of the speech recognizer. The runtimes are expressed either in absolute terms as durations or alternatively, in relative terms as real-time factors (xRT) i.e. the ratio of the recognition runtime to the length of the acoustic signal:

$$\text{xRT} = \frac{runtime(recognition)}{length(signal)} \tag{1.14}$$

Runtimes can however paint a somewhat distorted picture when the measuring conditions are not meticulously controlled. In an ideal world, every algorithm should be run on the same infrastructure, using the exact same set of machines which have no other active processes. In reality this is often impossible and it is for this reason that we repeat our measurements multiple times on the same machine and take the average over all the different measurements. If the algorithms are run on multiple machines and it is not possible to control which machines are being used, we will complement the reported runtimes with a complexity analysis as further proof of the efficiency of the algorithms.

1.6 Thesis Overview

In this thesis, we will discuss five different topics in as many chapters. A brief summary of each chapter can be found below.

- **Chapter 2: Compound-Head clustering**

  When it comes to efficiency, n-gram language models are second to none. For this reason we propose a technique that stays within the n-gram paradigm, but aggregates statistics over similar words in order to acquire more reliable estimates for rare words. The similarity criterion that we employ, is the one between a compound and its head. We describe a technique that finds this head with high precision, clusters the compound with its head and uses aggregated statistics to model unseen compounds as well as improve the modeling of infrequent compounds, at the same efficiency as a regular n-gram language model.

  This chapter was based on the following publications:

  - Joris Pelemans, Kris Demuynck, Hugo Van hamme and Patrick Wambacq. Coping with Language Data Sparsity: Semantic Head Mapping for Compound Words. In Proc. ICASSP, pages 141-145, Firenze, Italy, May 2014.
  - Joris Pelemans, Kris Demuynck, Hugo Van hamme and Patrick Wambacq. Improving N-gram Probabilities by Compound-head Clustering. In Proc. ICASSP, pages 5221-5225, Brisbane, Australia, April 2015.

- **Chapter 3: Semantic Language Models**

  Arguably the greatest weakness of n-gram models is their inability to capture long-distance relations between words. Semantic language models on the other hand exhibit opposite properties: they sacrifice word

order to be able to relate words by their meanings. In an attempt to have the best of both worlds, we propose the linear interpolation of n-grams with a state-of-the-art word similarity model which we converted into a semantic language model. Not only does this model outperform existing semantic models, it is also significantly more efficient.

This chapter was based on the following publication:

Joris Pelemans, Kris Demuynck, Hugo Van hamme and Patrick Wambacq. The effect of word similarity on N-gram language models in Northern and Southern Dutch. CLIN Journal, volume 4, pages 91-104, December 2014.

- **Chapter 4: Sparse Non-negative Matrix Language Modeling**

  The main reason why more recent and powerful models are inefficient is that they are exponential which makes normalization expensive. In this chapter, we investigate a new model, called Sparse Non-negative Matrix language model which is linear, hence does not suffer from this normalization issue. The model uses exponential meta-feature estimation and binary predictors to speed up training, and when trained with skip-gram features it performs as well as neural networks even though it takes 10x less computing power. Moreover, its ability to incorporate arbitrary features and adapt to given data makes it a very attractive model with many applications in fields as diverse as speech recognition and machine translation.

  This chapter was based on the following publications:

  – Joris Pelemans, Noam Shazeer and Ciprian Chelba. Sparse Non-negative Matrix Language Modeling. Transactions of the Association for Computational Linguistics, volume 4, pages 329-342, July 2016.

  – Joris Pelemans, Noam Shazeer and Ciprian Chelba. Pruning Sparse Non-negative Matrix N-gram Language Models. In Proc. Interspeech, pages 1433-1437, Dresden, Germany, September 2015.

  – Noam Shazeer, Joris Pelemans and Ciprian Chelba. Sparse Non-negative Matrix Language Modeling For Skip-grams. In Proc. Interspeech, pages 1428-1432, Dresden, Germany, September 2015.

- **Chapter 5: Layered Decoding for Large Vocabularies**

  When it comes to applying language models in automatic speech recognition, one obstacle that impedes the adoption of more complex models is the fact that most speech decoders combine all their knowledge sources into one huge search space. This makes for a complex task which means that each of the knowledge sources, including the language

model, have to be kept comparatively simple. In this chapter, we present *FLaVoR*, an alternative, layered architecture that uses the output of a first acoustic layer as input for a second word decoding layer. This decoupling alleviates the task of the decoder which makes it possible to apply more complex language models.

This chapter was based on the following publication:

Joris Pelemans, Kris Demuynck and Patrick Wambacq. A Layered Approach for Dutch Large Vocabulary Continuous Speech Recognition. In Proc. ICASSP, pages 4421-4424, Kyoto, Japan, March 2012.

- **Chapter 6: Language Model Adaptation for ASR of Spoken Translations**

  A recent study has shown that translators correcting the output of a machine translation system can be more efficient if they operate by voice. Rather than using an off-the-shelve ASR system, we investigate whether we can tune the recognizer by automatically adapting the language model to the source language text in such a way that the adaptation does not introduce too much overhead. We propose a direct integration technique that adapts the language model on a sentence-by-sentence basis, using word-level and phrase-level translation model probabilities of the source language text. By doing away with normalization and minimizing the amount of updates, we are able to achieve a substantial word error rate reduction on spoken Dutch translations from English, while introducing little to no overhead.

  This chapter was based on the following publications:

  – Joris Pelemans, Tom Vanallemeersch, Kris Demuynck, Hugo Van hamme and Patrick Wambacq. Efficient Language Model Adaptation for Automatic Speech Recognition of Spoken Translations. In Proc. Interspeech, pages 2262-2266, Dresden, Germany, September 2015.

  – Joris Pelemans, Tom Vanallemeersch, Lyan Verwimp, Kris Demuynck, Hugo Van hamme and Patrick Wambacq. Language Model Adaptation for ASR of Spoken Translations using Phrase-based Translation Models and Named Entity Models. In Proc. ICASSP, pages 5985-5989, Shanghai, China, March 2016.

- **Chapter 7: Conclusion**

  This chapter concludes the thesis by listing the original contributions and directions for future research.

Chapter 2

Compound-Head Clustering

Although n-grams are the most efficient and therefore still the most popular language models (LMs) in automatic speech recognition (ASR), they have two apparent disadvantages: first of all, they only operate locally and hence cannot model long-span phenomena such as sentence-wide or document-wide semantic relations. As we shall see in Chapters 3 and 4, this can be partly alleviated by extending n-grams with models or features that leverage longer distances, but continues to be a challenging research task as these extensions often come at the cost of a large increase in complexity.

The second disadvantage is caused by data sparsity: there is not enough training material to derive reliable count statistics for every possible (spoken) word sequence of length n, especially when n is large. Many word sequences and even single words only occur a limited number of times in the training material while others don't occur at all. Most language modeling research in the 90s was therefore focused on smoothing techniques that redistribute the probability mass and put aside some of the mass for unseen events [17,56,61,113]. Although smoothing does help, it doesn't overcome the actual flaw which is that n-grams generalize poorly to related words.

A more versatile approach was suggested by Brown et al. [12] who assign words to classes, each word in a class having similar properties. Instead of computing n-gram probabilities directly at the word level, count statistics are aggregated over all members of a class to achieve a higher level of abstraction, thereby reducing data sparsity. The advantage of this approach is that it allows generalization within the efficient paradigm of n-gram language modeling. Although this is promising in theory, it introduces a new and far from trivial problem of clustering words into classes. Indeed, for the idea of class n-

grams to work, the words in a class should ideally be mutually substitutable, i.e. they should be both semantically and syntactically similar. This is a challenging task and even if it is accomplished successfully it may still suffer from overgeneralization because of the many senses words can have [51]. In addition, most clustering algorithms rely on external knowledge such as a taxonomy or corpus statistics, where rare words are often underrepresented or not represented at all.

In this chapter, we present a novel clustering technique that alleviates the above issues for compound words. By mapping compounds onto their heads, the technique is able to estimate probabilities for new n-grams containing previously unseen compounds. It can also be applied in a more general fashion to improve probability estimates of existing n-grams. We argue that compounds are well represented by their heads which reduces the risk of overgeneralization and that retrieving the head is often possible without the need for external knowledge which allows the clustering of rare words. The technique is evaluated on Dutch read speech, but the idea may extend to languages with similar compound formation rules.

The chapter is organized as follows: Section 2.1 gives a linguistic description of compounds, zooms in on compounding in Dutch and discusses how compounds are usually modeled. In Sections 2.2 and 2.3, we present the main clustering algorithm and some extensions. Section 2.4 handles the integration of the compound-head clusters into the LM. Finally, Sections 2.5 and 2.6 validate the merits of the technique experimentally. We end with a conclusion in Section 2.7.

2.1 Compound Words

2.1.1 Linguistic Description

Compounding is the process of word formation which combines two or more words into a new word e.g. *foot+ball*. This should not be confused with derivation[1] where a word is combined with an affix instead of another word e.g. *recreation+al*. Compound formation rules vary widely across language types. This section is not meant to give an exhaustive overview, but rather to introduce the concepts that are relevant to our approach. Examples are limited to Germanic and Romance languages with which we are most familiar.

[1] Compounding and derivation are not the only word formation processes, but they are by far the most productive.

The manner in which two or more compound constituents are combined differs from language to language. In some languages such as English the constituents are mostly separated by a blank space. Others apply concatenation, possibly with the insertion of a binding morpheme. Still others use prepositional phrases to describe a relation between the head and the modifier e.g. the Spanish *zumo de naranja (lit: juice of orange)* or the French *machine à laver (lit: machine to wash)*.

Compounds can be broadly classified into 4 groups, based on their constituent semantics:

1. **endocentric compounds** consist of a head and modifiers which introduce a hyponym-hypernym or type-of relation e.g. *energy drink*.
2. **copulative compounds** have two heads, both of which contribute to the total meaning of the compound e.g. *sleepwalk*.
3. **appositional compounds** consist of two (contrary) classifying attributes e.g. *actor-director*.
4. **exocentric compounds** have a meaning that cannot be transparently derived from its constituent parts e.g. *skinhead*.

The position of the head also varies among languages and often corresponds to a specific manner of constituent combination. Germanic languages predominantly use concatenation with the head taking the rightmost position in the compound. Romance languages on the other hand are typically left-headed, applying the prepositional scheme mentioned above on the right-hand side.

In what follows we will focus on compounds in Dutch, which is our native language and the target language in our experiments, but we believe that the presented ideas extend to other languages on the condition that, like Dutch, they have a lexical morphology with concatenative and right-headed compounding.

2.1.2 Dutch Compounding

Like most Germanic languages, Dutch is a language with a relatively rich lexical morphology in the sense that new (compound) words can be made by concatenating two or more existing words e.g. *voor+deur+klink = voordeurklink (front door handle)*. Often the words are not simply concatenated, but separated by a binding morpheme which expresses a possessive relation between

the constituents or facilitates the pronunciation or readability of the compound e.g. *tijd+s+druk = tijdsdruk (time pressure)*. The majority of compounds in Dutch are right-headed and endocentric; some are copulative or appositional and a minority is exocentric [8]. Left-headed compounds do occur e.g. *kabinet-Vandeurzen (cabinet [of minister] Vandeurzen)*, but are rare.

2.1.3 Compound Modeling

In many languages, compounding is a productive process which induces the frequent creation of numerous new words all over the world. This process results in observing many compound words, most of them occurring rarely or with low frequency. As a consequence, these words are often not included in an n-gram LM or are included with a very unreliable probability. Moreover, even if sufficient training data is available, a typical application is limited in the number of words it can include in its vocabulary. These issues give rise to challenging problems in speech and language research which have been addressed by several authors for languages as diverse as German [79], Mandarin [117], and Hindi [25].

The most popular approach to address compounds in Dutch (and also in other languages) is to split them into their constituent parts and add these to the lexicon and LM. After recognition, the constituents are then recombined. Earlier research based on rule-based [66] and data-driven decompounding [82, 94] has shown that this does indeed reduce the word error rate (WER) for Dutch ASR.

However, this approach is mainly used to achieve maximal coverage with minimal vocabulary and has several disadvantages with regards to language modeling: (1) recompounding the emerging constituents is not trivial because many constituent pairs also exist as word pairs e.g. *winkelbediende (shopping assistant)* vs. *winkel bediende (shop assisted)*; (2) constituents of unseen compounds have never occurred together, which means that the LM can only make an unreliable decision based on unigram probabilities; and (3) given that in Dutch compounds, the first constituents generally play the role of modifiers while the last constituent acts as head of the compound [8], the left-to-right conditioning of probabilities in n-grams is a bad fit to the underlying principle.

Although our proposed approach also employs decompounding, it is important to note that it is substantially different from the large number of algorithms performing lexicon reduction. Instead, we use the decompounding information to introduce new knowledge into the LM in order to model compounds when data is scarce. This overcomes the language modeling issues mentioned above and allows us to extend the vocabulary with new, unseen words as well as to

improve probability estimation for already observed words. It does not however reduce the size of the vocabulary.

2.2 Compound-Head Clustering

2.2.1 Word Clustering

Word clustering approaches can be divided into two classes: knowledge-driven and data-driven. Knowledge-driven approaches make use of some external knowledge sources that contain information about words which facilitates clustering. The knowledge can be syntactic e.g. part-of-speech tags or semantic e.g. synonym sets in WordNet. Although this knowledge can be a valuable source of information, it is typically the result of manual labor which means that it is often incomplete or even non-existent and that annotators don't always agree or even make mistakes.

In data-driven approaches, words are clustered using statistics that are computed on a large pool of data. The statistics are then used to optimize a criterion such as maximum likelihood or mutual information. Although these approaches do not depend on human annotators, they are not without problems. There is only so much data and it is hard to find reliable clusters for words that do not occur often in the data. And it is exactly these infrequent words that would benefit the most from clustering, as probability estimates for more frequent words are already more reliable.

Both approaches have the additional disadvantage of overgeneralization. Although the created clusters contain similar words, many words also have multiple senses and as such exhibit different behavior in different contexts.

2.2.2 Compound Clustering

We believe that the issues mentioned above are less problematic for compounds when they are clustered based on their head. First, with regards to overgeneralization, the clusters are more homogeneous as all of the members are well represented by their head, both syntactically and semantically. For most compound words, the head has the unique property of carrying inherent class information. This is obviously the case for the predominant class of endocentric compounds which introduce a hyponym-hypernym relation. It can be argued though that this is also true for copulative and appositional compounds. While these two types of compounds do not restrict the meaning of the compound,

their heads can still be viewed as classes. The only troublesome compounds are exocentric compounds. However, because of their opaque meaning, they are in fact quite rare.

Second, by mapping a compound onto its head we effectively apply a clustering that does not depend on external information and can hence be applied to all compounds, regardless of their frequency in a training corpus. This compound-head clustering (CHC) simplifies introducing new words which opens up possibilities for domain adaptation. To our knowledge, this approach has not been described in the literature for any language and is substantially different from the mentioned decompound-recompound approaches [66, 79, 82, 94] that fail to take advantage of the valuable semantic information embedded in compounds.

To obtain heads for compounds, one could make use of existing knowledge sources such as lists of compounds that were manually split into their constituents. However, upon investigation this type of knowledge sources proved to be insufficient for our needs, mostly because a head can consist of more than one constituent. In addition, no morphological information is available for infrequent compounds, which are the main target of our technique.

In the following sections, we therefore propose a clustering algorithm consisting of 2 parts: (1) a generation module which generates all possible decompounding hypotheses; and (2) a selection module which selects the most plausible head.

2.2.3 Generation Module

First, all possible decompounding hypotheses are generated by means of a brute-force lexicon lookup: for all possible substrings w_1 and w_2 of the candidate compound w, $w = w_1 + w_2$ is an acceptable hypothesis if w_1 and w_2 are both in the lexicon. The substrings are optionally separated by the Dutch binding morphemes 's', 'en' or '-'. The module also works recursively on the first substring i.e. if w_1 is not in the lexicon, the module will verify whether or not it's a compound itself. In its current implementation, the system always makes the assumption that the head is located at the right-hand side of the compound, since this is almost exclusively the case for Dutch, as we discussed in Section 2.1.2. Hence, we do not expect this assumption to significantly influence the results.

We hypothesize that there is a significant discrepancy between the frequency of compound modifiers and heads: since a (endocentric) compound is typically a hyponym of its head and most if not all hypernyms have multiple hyponyms, the heads tend to occur frequently. Modifiers on the other hand are less frequent,

because they constrain the hypernym to a more specific and often completely new domain e.g. *schaak+stuk (chess piece)*. To account for this discrepancy we allow the generation module to read from 2 different lexica: a modifier lexicon V_m and a head lexicon V_h. Although the 2 lexica can be filtered in any way, the main implementation only adopts word frequency filters. An exception is made for acronym modifiers consisting exclusively of uppercase characters or an uppercase character followed by digits, which are automatically considered as valid modifiers and are therefore not required to be lexical e.g. *NAVO-verdrag (NATO treaty)* or *F1-piloot (F1 pilot)*.

We further expect the amount of (false) hypotheses to increase drastically with decreasing constituent length which is especially true if the lexica contain (noisy) infrequent short words. Two parameters L_m and L_h are introduced to control the minimal length of modifiers and heads respectively.

2.2.4 Selection Module

The generation module hugely overgenerates because it only has access to lexical knowledge. In the selection module we introduce knowledge based on corpus statistics to select the most likely candidate. Concretely, the selection between the remaining hypotheses is based on unigram probabilities and constituent length. We expect longer and more frequent constituents to yield more accurate results and provide selection parameters w_{len}, w_u and w_{pu} to weigh the relative importance of the head length, head unigram probability and product of the constituent unigram probabilities. We also considered the use of part-of-speech (POS) knowledge, but did not achieve any improvements with it, most likely due to incorrect POS tagging of the infrequent compounds.

Figure 2.1 shows pseudocode for the complete CHC algorithm, excluding the constituent separation by binding morphemes for the sake of clarity.

2.3 Extensions for Observed Compounds

Until now we have presented a general algorithm that can be applied to any compound regardless of its frequency. In [84], where we wanted to introduce new compounds to the language model, we observed that it is hard if not impossible to improve on this basic algorithm, because we have little or no information about these compounds. However, in [85] where we targeted more frequent compounds, we were able to use additional knowledge to improve upon

function GENERATE($compound, V_m, V_h, L_m, L_h$)
  $hypotheses \leftarrow \{\}$
  for all $mod + head = compound$ **do**
    if $len(mod) \geq L_m$ **and** $len(head) \geq L_h$ **then**
      if $head \in V_h$ **then**
        if $mod \in V_m$ **or** *mod is acronym* **then**
          $hypotheses.add(mod, head)$
        else
          $hypotheses.add(\text{GENERATE}(mod, ...), head)$
  return *hypotheses*
function SELECT_BEST($hypotheses, w_{len}, w_u, w_{pu}$)
  $max\_score \leftarrow 0$
  for all $(mod, head) \in hypotheses$ **do**
    $score \leftarrow w_{len} * length(head) + w_u * P_{uni}(head)$
      $+w_{pu} * P_{uni}(mod) * P_{uni}(head)$
    if $score > max\_score$ **then**
      $max\_score \leftarrow score$
      $best \leftarrow (mod, head)$
  return *best*

Figure 2.1: Compound-head clustering algorithm.

the basic algorithm. In this section, we explain the different extensions that we introduced to exploit this additional knowledge.

2.3.1 Word-affix Disambiguation

Dutch has a number of letter sequences that can act both as a word and as an affix, most of which are relatively short. The CHC algorithm we described in Section 2.2 cannot distinguish between the two, since it has no access to semantic knowledge. This leads to a number of errors where a prefix is interpreted as a modifier e.g. *ver+lengen* (*far+lengthen*) instead of *verlengen* (*lengthen*); or where a suffix is interpreted as a head e.g. *gevaar+lijk* (*danger+corpse*) instead of *gevaarlijk* (*dangerous*). Although in many cases this phenomenon can be prevented by increasing the minimal length of the modifiers/heads, this also lowers the system's recall. Better solutions are at hand, without the need for semantic ontologies. To address this issue, we made lists of all the sequences of k letters at the start and end of each word in our corpus and counted how many times they occurred as a word or as an affix. The idea is that affixes perform a syntactic role that can be applied to

a large number of words and are therefore more likely to occur than regular words. Letter sequences that occur significantly less as a word than as an affix, are then assumed to be prefixes/suffixes and are subsequently removed from the modifier/head lexica in the generation module. This solution also has the positive side effect that some spelling mistakes and foreign heads are filtered out. In what follows, the extent of affix-word ambiguity handling is indicated by the amount of modifiers a_m and heads a_h that were removed from the constituent lexica.

2.3.2 Conservative Clustering

In [84], we only clustered unseen words. As such, clustering errors are not likely to be costly: if a non-compound is clustered, or if a compound is mapped to the wrong head, it is unlikely that this will do much damage to the speech recognition. The unseen word will most likely not be recognized, which was already the case before we introduced it and the probability mass of the head will only be reduced by a small amount (see Section 2.4.2).

In the case of the more general approach however, incorrect mappings may be catastrophical. Because the amount of words that undergo clustering is potentially much larger, the probability mass of the hypothesized head may be severely reduced. Moreover, many words that were correctly recognized before, may no longer be recognized. Clearly, in this scenario, clustering should be undertaken with more caution.

This caution was implemented by attributing a larger relative importance to precision than to recall during optimization. We achieved this by assessing our clusters with the F_β-score which is well known in the field of information retrieval:

$$F_\beta = (1 + \beta^2).\frac{precision.recall}{(\beta^2.precision) + recall} \tag{2.1}$$

where β indicates the weighting ratio of recall and precision. In our case, β should thus be lower than 1.

2.3.3 Morphological Knowledge

In [84], we concluded that the use of POS information did not yield any improvement to the clustering of unseen compounds. This was mostly caused by incorrect POS tags assigned to both the infrequent compounds and to a lesser extent also the constituents. The data sparsity problem had thus shifted from language modeling to POS tagging. When we target more frequent compounds,

this is not the case. The clustering is less hampered by sparsity and is helped by the inclusion of sophisticated linguistic knowledge based on POS rules.

As a first rule, we force the compound and its head to have identical POS tags, since a change in POS tag is unlikely to correspond to a correct head. This compound-head POS constraint requires the POS tagger to be fine-grained in order to distinguish between e.g. noun genders or verb tenses. To further constrain the search space we apply compounding rules that only allow combining constituents with certain POS tags e.g. concatenating two nouns is productive, hence allowed, but concatenating two verbs is not. These rules are implemented based on [110] where the author showed that they severely restrict overgeneration.

As a final improvement, we investigate the effect of a lexical database that contains manual morphological annotations of a large amount of words, which we expect to be beneficial for both recall and precision.

2.4 Language Model Integration

In this section, we describe how the acquired clusters were integrated into the language model. Although compound-head clusters can be useful in different models, we opted for class-based n-gram models, because of their efficiency.

2.4.1 Class-based Models

The proposed technique is inspired by class-based n-gram models, as introduced by Brown et al. [12]. The idea of class n-grams is that words can be similar to others in their meaning and syntactic function. Grouping such words into classes can help overcome the data sparsity in training material, since the prediction of infrequent or unseen words is then based on the behavior of similar words that have been seen (more often). Equation (2.2) shows how the n-gram probabilities are computed:

$$P(w_k|w_1^{k-1}) = P(C_k|C_1^{k-1})P(w_k|C_k) \tag{2.2}$$

where w_k and C_k denote the word and class at position k respectively and w_1^{k-1} and C_1^{k-1} denote the word and class sequences from positions 1 to $k-1$.

Class-based n-gram models are interesting, because they allow generalization within the efficient paradigm of n-gram language modeling. A problem with class-based approaches however is that they tend to overgeneralize: the

hypothesis that all words in the same class behave in a similar fashion is too strong. This was certainly true for [12] where no perplexity reduction was observed. However, as we mentioned in Section 2.2.2, we believe that our approach should suffer less from overgeneralization, since compounds are well represented by their heads. As we will see in Section 2.6, this will allow us to do large scale clustering with positive results.

2.4.2 Probability Estimation

To integrate the compound-head clusters into the class-based paradigm we need to estimate two probabilities as given by Eq. (2.2): the transition probability $P(C_k|C_1^{k-1})$ and the emission probability $P(w_k|C_k)$.

For already observed compounds these two probabilities are easy to compute. The computation of the transition probability is analogous to regular word-based n-grams, where instead of counting the individual words, the counts are now aggregated over the entire class. The emission probability is then computed by the relative frequency of each compound w in the class:

$$P(w|C_{head}) = \frac{c(w)}{c(head) + \sum_{w' \in C_{head}} c(w')} \tag{2.3}$$

where C_{head} denotes the class defined by the head.

The compound-head clusters can also be used to enrich a language model with probability estimates for new, unseen compounds. Since we have argued that a compound is well represented by its head, we use the n-gram probability of the head as the transition probability for each member. The emission probability can be estimated by replacing the count $c(w)$ in Eq. (2.3) with an estimate $\hat{c}(u)$ for each unseen compound u. A sensible value for $\hat{c}(u)$ can be obtained empirically or more analytically, by averaging over the counts of all cut-off out-of-vocabulary (OOV) compounds with the same head i.e. the least frequent compounds with the same head which are cut off or disregarded during LM training. An alternative approach consists of distributing the probability mass uniformly within each class.

2.5 Experiment 1: Introducing New Words

In this first experiment, we want to investigate whether compound-head clusters can be used to improve the modeling of previously unseen compounds.

New compounds are added to the language model and different techniques to estimate n-gram probabilities for these compounds are compared.

2.5.1 Setup

Our LM training data consists of a collection of normalized newspaper texts from the Flemish digital press database Mediargus which contains 1104M word instances (tokens) and 5M unique words (types) from which we extracted all the mentioned vocabularies and word frequencies. Vocabularies of V words always contain the V most frequent words in Mediargus. They were converted into phonemic lexica using an updated version of [29] and integrated, together with the created LMs, into the recognizer described in [31]. The development data for the CHC originates from CELEX [3] where the ground truth is based on a morphological analysis of 122k types of which 68k are compounds. For each compound only one possible head is allowed which is optimal for most compounds, but might be too strict for others e.g. *veen+bessen+sap (cranberry juice)* should be mapped to the semantically most similar head *bessensap (berry juice)*, but a mapping to *sap (juice)* is still acceptable. The test data consists of the Flemish part of the Corpus Spoken Dutch [81] or *CGN* component o, which contains read speech. In order to focus on the efficiency of our proposed technique, the component was reduced to those fragments that contain unseen compounds for which a head was retrieved. This was achieved by comparing the reference transcription with the recognizer's vocabulary, manually splitting the compounds that were not in the vocabulary and removing those fragments that did not contain any of these compounds. After reduction, the test data which we will further refer to as CGN-o, contains almost 22h of speech. It consists of 192,153 tokens, produced by 25,744 types of which 1,625 are unseen in the LM training data and 953 are compounds.

2.5.2 Clustering

We applied an extensive grid search on the CELEX development data for all of the system parameters and counted the amount of true and false positives and negatives. We then calculated the precision and recall for each parameter setting and found that the optimal results were achieved with V_m=600k, V_h=200k, L_m=3, L_h=4, w_{len}=1, w_u=0 and w_{pu}=0. Table 2.1 shows that these parameters yield a precision of 80.31% and recall of 82.01% on the development data. When tested on the evaluation set, the precision is roughly equal with 80.25%, but the recall is even better with 85.97%. Moreover, many of the mappings that do not correspond to the ground truth are similar to the

CELEX (dev)		CGN-o (eval)	
precision	recall	precision	recall
80.31	82.01	80.25	85.97

Table 2.1: Clustering results as measured by precision and recall on CELEX and CGN-o.

veenbessensap example. Although these mappings are suboptimal, they are nonetheless adequate, hence likely to have a positive impact on the LM.

2.5.3 Language Model Integration

We trained initial, open vocabulary n-gram LMs of orders 2 to 5 with modified Kneser-Ney smoothing on the 400k most frequent words in Mediargus. The remaining, cut-off OOV words were used to gather statistics for unseen words in a general OOV class. We then expanded the 400k vocabulary with the unseen compounds for which the CHC algorithm found a head. This new, expanded vocabulary was used when comparing WERs for the different estimation techniques.

As a baseline we considered two techniques that do not have the compound-head clusters at their disposal. Hence, these techniques have to resort to general OOV statistics i.e. the probability mass for the OOV class is redistributed over the newly added compounds using Eq. (2.3), where all compounds are clustered in a single OOV class instead of their respective head clusters. The redistribution was done in two ways: uniformly and, analogous to Section 2.4.2, based on the average cut-off OOV unigram count of all the compounds with the same head.

OOV clustering was compared to both the unigram-based and uniform CHC approaches, discussed in Section 2.4.2. Although we also attempted to optimize the count estimate $\hat{c}(u)$ for unseen word u empirically for both OOV and compound-head clustering, these results are not reported, as they did not invariably improve the results for all n-gram orders.

Table 2.2 shows the WERs of all these approaches, compared to the WERs of the initial LMs with 400k words, where no clustering was done. As can be seen, OOV clustering performs surprisingly well with regards to the initial LMs which seems to indicate that lexicon expansion is sufficient to recognize most of the unseen compounds. We suspect that this is due to the nature of our test set, which contains clean, read speech, and we expect this effect to be smaller with a more challenging data set. Unexpectedly, unigram-based OOV clustering

	n-gram order			
clustering technique	2	3	4	5
no clustering	31.31	28.23	27.59	27.53
uniform OOV clustering	30.70	27.67	27.02	26.97
unigram-based OOV clustering	30.63	27.59	26.96	26.90
unigram-based CHC	30.69	27.65	27.00	26.95
uniform CHC	**30.33**	**27.29**	**26.65**	**26.62**
oracle experiment	30.20	27.14	26.51	26.48

Table 2.2: WERs for the initial 400k words LMs (no clustering) and the different clustering techniques, as calculated on CGN-o. The results for the oracle experiment were acquired by automatically correcting the compound recognition errors of the uniform CHC setup.

also performs better than unigram-based CHC. Upon further investigation, we found that this was not caused by a low CHC n-gram coverage, but by an underestimation of $\hat{c}(u)$ due to the low counts of the cut-off OOV compounds, compared to the count of their heads. This shows that the unigram-based estimator is not reliable, as it is too dependent on the otherwise unused cut-off LM training data. The results for uniform CHC confirm this conclusion, as they produce a significant (Sign and Wilcoxon test $p < 0.0001$) relative WER reduction of approximately 1% over OOV clustering. This performance improvement is more or less constant over the different n-gram orders and, as it turns out, is also very close to that of an oracle experiment in which the remaining compound recognition errors were automatically corrected.

2.6 Experiment 2: In-vocabulary Modeling

In this experiment, we want to examine whether and to what extent compound-head clusters suffer from overgeneralization. We do this by exposing different fractions of the vocabulary to the CHC algorithm which tells us whether more frequent compounds can be clustered without harming the language model.

2.6.1 Setup

The LM training data and the recognizer for this experiment are the same as in Section 2.5. The CHC algorithm was extended with the adaptations mentioned in Section 2.3. The POS tags, used for the morphological rules, were generated by running the Dutch POS tagger Frog [109] on the entire LM training data,

after which only the most frequent POS tag was kept for each word. The lexical database from which we extracted decompoundings is CELEX [3] which contains morphological analyses of 122k types of which 68k are compounds.

The ASR development and test data each consist of 200 fragments of the Flemish part of the Corpus Spoken Dutch [81] component o, which contains read speech. Both of them contain around 6.5h of speech and consist of about 60k tokens, produced by 10k types. As opposed to the experiments presented in Section 2.5, the data sets were not reduced to fragments that only contain compounds, to show that clustering of in-vocabulary words does not yield worse LM statistics and hence higher WERs for non-compounds.

2.6.2 Clustering Improvements

Although the CHC system that was proposed in [84] did a fine job in handling unseen compounds, it would not be suited for large-scale application, because it has too many false positives. As we mentioned in Section 2.3.2, the damage done by an incorrect mapping is likely to be larger than the positive impact of a correct mapping, so only a system that attaches more importance to precision than to recall would be suited to our purposes. The F-score was derived so that F_β measures the accuracy of a system that attaches β times as much importance to recall as to precision.

We did not attempt a thorough optimization of β, as it is only an indirect measure of the clustering quality. Instead, we performed ASR experiments with retrained class-based n-gram LMs, using the optimal clustering systems with regards to precision and recall and chose the one that performed best on the development data. Since ASR experiments are computationally expensive we then determined an optimal value of β for future experiments. Table 2.3 shows 5 different systems and their parameters. These are not the only systems that we investigated, but they were selected to compare the effect of the different parameters. Their precision, recall, WER and F-scores on the development data are shown in Table 2.4. Note that the WERs are achieved by feeding the 700k least frequent words in the 800k words vocabulary to the CHC algorithm (see Section 2.6.3).

In our initial attempts to reduce the number of false positives we found that increasing the minimum constituent length to 5 or 6, and pruning the most probable affixes, drastically decreased the recall and as such limited the potential of CHC_1 (P=88.5%, R=12.0%, WER=27.55%). This shows that more intelligent pruning is necessary and the POS rules as stated in [110] provide exactly this.

	V_m	V_h	L_m	L_h	a_m	a_h	rules	db
CHC_1	600k	200k	5	6	200	300	-	no
CHC_2	20k	20k	4	4	0	0	1-2	no
CHC_3	20k	20k	4	4	0	10k	1-2	no
CHC_4	20k	20k	4	4	0	10k	1-2	yes
CHC_5	200k	100k	3	4	0	0	1-5	yes

Table 2.3: Different CHC systems and their parameters. The columns indicate (from left to right) the generation parameters (V_m-a_h), which POS rules were employed, and whether or not the system had access to a lexical database.

	P	R	WER	$F_{0.2}$	$F_{0.5}$
CHC_1	88.5	12.0	27.55	71.07	38.9
CHC_2	90.6	25.0	27.25	82.29	59.42
CHC_3	**93.3**	21.9	27.14	82.90	56.48
CHC_4	89.6	31.3	27.15	**83.61**	65.28
CHC_5	80.2	**46.6**	**27.09**	78.04	**70.09**

Table 2.4: CHC fitness as measured by precision (P), recall (R), WER (with 700k least frequent words in the 800k words vocabulary fed to CHC) and F-scores on the development data.

The application of each rule separately shows that most of the compounds follow one of the following 2 rules:

1. noun + noun = noun e.g. *spraak+herkenning* (*speech recognition*)
2. verb stem + noun = noun e.g. *speel+plein* (*playground*)

When combined, these rules allow for a precise CHC setup (CHC_2) with better recall and lower WER (P=90.6%, R=25.0%, WER=27.25%), even when allowing relatively short constituents of length 4. Although this setup disallows all other possible decompoundings and thus employs a very rigorous pruning, it still benefits from affix-word ambiguity resolution: pruning the 10,000 most probable suffixes yields an absolute precision increase of 2.7% at the cost of a recall decrease of 3.1%, which lead to a further WER reduction of 0.11% (CHC_3).

In fact, when evaluated on the development set, this simple system turns out to be very competitive to systems that include external knowledge from a lexical database with manual morphological annotations.

Only CHC_5 is able to beat it by a small margin (P=80.2%, R=46.6%, WER=27.09%), employing these additional morphological rules:

3. adverb + noun = noun e.g. *onder+broek* (*underpants*)
4. noun + adj = adj e.g. *sneeuw+wit* (*snow white*)
5. quantifier + quantifier = quantifier e.g. *zes+tien* (*sixteen*)

Notice that CHC_4 is a nice illustration of the improvements achieved by the extensions described in Section 2.3. It has a precision that is comparable to (and even a bit higher than) CHC_1, but an increased recall of almost 20% absolute.

At first glance, it is not obvious what β one should employ for future experiments. However, if we distinguish between the setups that don't have access to the lexical database (CHC_1, CHC_2 and CHC_3) and those that do (CHC_4 and CHC_5), we find that, although both groups require emphasis on precision, the second group attributes more weight to recall ($\beta \geq 0.5$) than the first group ($\beta \leq 0.2$). This is logical, since with access to external morphological knowledge a system can afford to make more mistakes.

2.6.3 Clustering Degree

For a thorough investigation of the in-vocabulary CHC, we trained an open vocabulary word-based 3-gram LM with modified Kneser-Ney smoothing on the 800k most frequent words in Mediargus. Different fractions of the lexicon were fed to the CHC algorithm to investigate to what extent CHC would be useful for (in)frequent compounds. Fractions were chosen based on word (in)frequency e.g. a fraction of 50% indicates that the 400k least frequent words in the LM training data undergo decompounding. Only those that are judged to be a compound by the CHC system are clustered together with their head. For each fraction then, various CHC settings were explored and optimized on the development data, as was already explained in Section 2.6.2. We found that the optimal system corresponds to CHC_5 in Table 1, which shows that 0.5 is a reasonable value for β.

Table 2.5 gives an overview of the WERs on both the development and evaluation data for this optimal system. It can be seen that for all fractions, CHC constitutes a small, but significant improvement over the baseline word-based n-gram LM, which is indicated by a 0% fraction. Another thing to remark, is that the results continue to improve as the fraction undergoing CHC becomes larger, unless when CHC is applied to the complete lexicon. We suspect that this is due to some overgeneralization by clustering many frequent

	fraction of lexicon undergoing CHC					
	0%	50%	75%	87.5%	93.75%	100%
	0	400k	600k	700k	750k	800k
dev	27.70	27.19	27.12	27.09	**27.08**	27.31
eval	26.68	26.35	26.35	26.31	**26.27**	26.43

Table 2.5: WERs (in %) in function of the fraction of the 800k lexicon undergoing CHC, compared to a baseline (0%) 3-gram LM. Fractions correspond to the least frequent words in the LM training data.

compounds: a very frequent compound w_{freq} e.g. *wedstrijd* (*game*, lit. *gamble battle*) is more likely to have a unique meaning and context that is different from its head. If it is clustered together with the head and the potentially many compounds that have the same head, it will lose some of its probability mass to these other class members, which may be harmful for contexts that are unique to w_{freq}. It should however be noted that even in these extreme cases, CHC still outperforms the baseline.

The WER reductions are considerable, given that we are only trying to correct compound words. This becomes more clear when we check how many of the previously misrecognized compounds were actually corrected. In the scenario of feeding 93.75% or 750k words of the lexicon to the CHC algorithm, approximately 220k words or 27.5% of the lexicon are clustered of which only 592 types and 632 tokens actually occur in the test data. Of these, 75.7% of the types and 76.3% of the tokens were correctly recognized after integrating the compound-head clusters into the 3-gram LM. Other scenarios are similar. The fact that clustering more than a quarter of the complete vocabulary does not harm the WERs, but on the contrary, that after clustering, more than 75% of the compounds are correctly recognized, shows that the achieved improvements are substantial and indicates that CHC is a better way to model compounds in Dutch than classical word-based n-gram LMs.

2.7 Conclusion

We have introduced a new clustering technique to cope with language data sparsity by mapping compound words onto their heads. In a first experiment, we investigated whether compound-head clusters can be used to improve the modeling of previously unseen compounds. Results on Dutch read speech show that our technique is capable of correctly identifying compounds and their heads with a precision of 80.25% and a recall of 85.97%. A class-based language model

with compound-head clusters achieves a significant, relative reduction in WER of 1% at the same efficiency of a regular n-gram language model.

In a second experiment, we applied several extensions to the basic CHC algorithm which enables its application on in-vocabulary words. By including word-affix ambiguity filters, morphological rules based on part-of-speech and a lexical database with morphological analyses we created high precision compound-head clusters with increased recall e.g. an absolute recall increase of 20% for a system with 90% precision. We have proven empirically that the resulting compound-head clusters, when used in a class-based n-gram language model, are capable of estimating more reliable n-gram probabilities of observed compounds i.e. compounds for which training data is available. Moreover, we have shown that a conservative, but accurate clustering may be applied to a large fraction of the lexicon to achieve a compound recognition of more than 75% and a significant WER reduction, compared to a baseline word-based n-gram language model.

Chapter 3

Semantic Language Models

In the previous chapter, we proposed a clustering technique that can be incorporated into n-gram language models, thereby improving the model without increasing its complexity. However, n-grams have a fundamental flaw: their primary assumption that the history upon which the prediction is based, can be reduced to only a handful of words, is clearly incorrect. Although it may be the case that much if not most of the information resides in the immediate, local context of the current word [96], other long-span phenomena such as sentence-level or document-level semantic relations can only be modeled with the help of the more distant history. One of the ways to address this issue is to keep the n-gram language model (LM) to model local phenomena and combine it with *semantic language models* that model word similarity in the hope of also capturing more global phenomena.

The focus of this chapter is to examine the combination of n-gram LMs with several of these models including the well-established cache models [64] and language models based on Latent Semantic Analysis [5, 26]. We also propose a new semantic language model based on the recently proposed continuous skip-gram model [71, 73, 74], which is a scalable adaptation of a Neural Network Language Model (NNLM) [7] and the current state of the art in word similarity. We discuss the strengths and weaknesses of each of these combined models, based on their predictive power of the Dutch language and investigate whether and in what way the effect of Southern Dutch training material differs when evaluated on Northern and Southern Dutch material.

The chapter is organized as follows. In Section 3.1, we give an overview of several classical models, which serve as a baseline to compare our proposed model with. This model is based on the continuous skip-gram model and

is explained in more detail in Section 3.2. Section 3.3 covers an in-depth analysis of each of the models and compares their predictive power of the Dutch language. Finally, in Section 3.4, we present some thoughts on how the models could be integrated in the first pass of a speech recognizer. We end with a conclusion in Section 3.5.

3.1 Classical Models

3.1.1 Cache Models

Cache models are based on the observation that topical words tend to re-occur within a text. A cache memory is kept that keeps track of the last K words in a document and is consulted when predicting the next word. In their simplest form [64], cache models distribute the probability mass uniformly among all the K tokens in the cache memory:

$$P_{cache}(w_q|w_{q-K}^{q-1}) = \frac{C_{cache}(w_q)}{K} \tag{3.1}$$

where $C_{cache}(w_q)$ indicates the frequency of word w_q in the cache memory.

Extensions exist [21] where word age is taken into account i.e. the impact of older tokens is decreased by applying a (typically exponential) decay weighting function:

$$P_{cache}(w_q|w_{q-K}^{q-1}) = \beta \sum_{j=q-K}^{q-1} I(w_q = w_j)\alpha^{(q-j)} \tag{3.2}$$

where $I(x)$ is an indicator function that takes the value 1 if x is true and 0 otherwise, α is the decay rate and β is a normalization factor.

Cache models are simple and efficient and are capable of modeling long distance phenomena which is one of the main weaknesses of n-gram LMs. They do not however employ any kind of semantic knowledge.

3.1.2 Latent Semantic Analysis

Latent Semantic Analysis (LSA) [26] is one of the earliest attempts to discover hidden semantic structure in a text by considering word co-occurrences. It is a dimensionality reduction technique based on truncated Singular Value

Decomposition that is applied on a $m \times n$ term-document matrix $\mathbf{C}$ which contains in each cell the number of times a word (rows) occurs in a document (columns). As is shown in Figure 3.1, this count matrix is decomposed into a product of an orthogonal $m \times r$ matrix $\mathbf{U}$, a diagonal $r \times r$ matrix $\mathbf{S}$, containing the singular values, and another orthogonal $r \times n$ matrix $\mathbf{V}^T$. The information in $\mathbf{C}$ that corresponds to the smallest singular values is considered to be noise induced by data errors and word redundancy and is effectively removed by preserving only the k largest singular values. The resulting rank k approximation is optimal with regards to the Frobenius norm and uncovers latent semantic relations between words and documents.

Often a preprocessing step is useful, because the documents are not of equal length and not all words are equally informative. To this end, the raw counts of $\mathbf{C}$ may be transformed according to a weighting scheme which typically consists of a global component $G(i)$ and a local component $L(i, j)$ [38]:

$$C'(i, j) = G(i)L(i, j) \tag{3.3}$$

where $\mathbf{C}'$ represents the count matrix $\mathbf{C}$ after applying the weighting scheme. The same global weight – indicating the overall importance of a term – is applied to an entire row of the matrix, whereas the local weight – which indicates the importance of a term in a specific document – is applied to each cell in the matrix.

Although many different schemes exist, the choice is not so critical to the overall performance of LSA and in this work we only investigate the use of *term frequency - inverse document frequency* (TF-IDF), which is one of the most used weighting schemes in the context of information retrieval. For an overview and classification of different weighting schemes, we refer to [99].

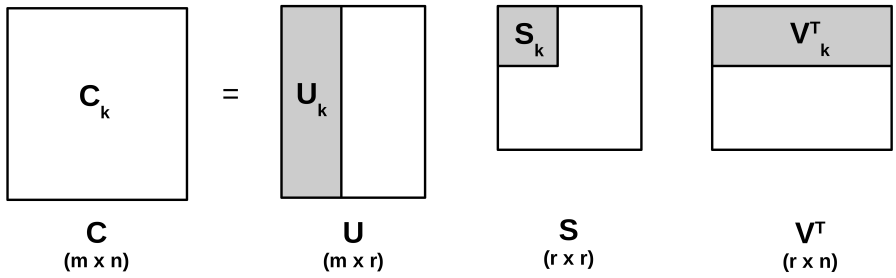

Figure 3.1: (Truncated) Singular Value Decomposition.

Semantic similarity between documents and words is measured by computing the cosine similarity between their vectors, projected in the latent space. Since the relation between a given word w_i and document d_j is expressed by the cell C_{ij} of the (weighted) count matrix $\mathbf{C}$, it can be easily shown that their projections in the latent space correspond to $\mathbf{u_i S}^{1/2}$ and $\mathbf{v_j S}^{1/2}$, respectively, where the indices i and j indicate the columns in $\mathbf{U}$ and $\mathbf{V}$ that correspond to the word and the document. The cosine similarity can then be computed as follows:

$$\cos(\mathbf{u_i S}^{1/2}, \mathbf{v_j S}^{1/2}) = \frac{\mathbf{u_i S v_j}^T}{\|\mathbf{u_i S}^{1/2}\| \|\mathbf{v_j S}^{1/2}\|} \tag{3.4}$$

In the context of language modeling, the history is considered to be a (pseudo-)document $\tilde{d}_{q-1}$ and its projection in the latent space is represented by $\tilde{\mathbf{v}}_{\mathbf{q-1}}$. Using the short-hand notation $D(u_i, v_j) = \cos(\mathbf{u_i S}^{1/2}, \mathbf{v_j S}^{1/2})$, the cosine similarity between the projections of the target word $\mathbf{u_q}$ and the history $\tilde{\mathbf{v}}_{\mathbf{q-1}}$ can then be converted into a probability as follows [22]:

$$P(w_q|\tilde{d}_{q-1}) = \frac{[D(\mathbf{u_q}, \tilde{\mathbf{v}}_{\mathbf{q-1}}) - \min_{\mathbf{u}} D(\mathbf{u}, \tilde{\mathbf{v}}_{\mathbf{q-1}})]^\gamma}{\sum_{w_i \in \mathcal{V}} [D(\mathbf{u_i}, \tilde{\mathbf{v}}_{\mathbf{q-1}}) - \min_{\mathbf{u}} D(\mathbf{u}, \tilde{\mathbf{v}}_{\mathbf{q-1}})]^\gamma} \tag{3.5}$$

where $\mathcal{V}$ is the vocabulary and γ is a parameter that controls the dynamic range of the distribution.

Although LSA is capable of uncovering semantic relations between words and documents, it is insensitive to the multiple senses that many words have. Moreover, using the Frobenius norm as an error function assumes normally distributed data with independent entries, which is not the case for the counts in the term-document matrix.

3.2 Continuous Skip-gram Model

3.2.1 Background

Motivated by the successes in neural network language modeling, Mikolov et al. [71, 73, 74] recently proposed an architecture for the acquisition of high-quality continuous word vectors or *word embeddings*. This architecture might not be able to represent the data as precisely as a recurrent neural network, but is less complex and can therefore handle more data efficiently. The continuous skip-gram model (CSM) is the current state of the art in word similarity and

is shown in Figure 3.2. It is a log-linear classifier that predicts output words w_O within a range R before and after an input word $w_I = w(t)$ at time t.

First, it projects the input word onto a continuous layer with weights $\mathbf{H}$:

$$\mathbf{h} = \mathbf{x}^\mathbf{T}\mathbf{H} \tag{3.6}$$

As the input word w_I is represented by a one-hot encoding $\mathbf{x}$ of the vocabulary $\mathcal{V}$, this projection essentially boils down to selecting the row in $\mathbf{H}$ that corresponds to w_I. In what follows, we denote this row vector by $\mathbf{v}^\mathbf{T}_{\mathbf{w_I}}$ – the word embedding for word w_I. To ensure that the output layer forms a valid probability distribution over the vocabulary $\mathcal{V}$, the model then uses a softmax activation function with weights $\mathbf{W}$. If we denote the column in $\mathbf{W}$ that corresponds to the softmax weights for output word w_O by $\mathbf{v'_{w_O}}$, the model can be represented as follows:

$$P(w_O|w_I) = \frac{\exp(\mathbf{v}^\mathbf{T}_{\mathbf{w_I}}\mathbf{v'_{w_O}})}{\sum_{w\in\mathcal{V}}\exp(\mathbf{v}^\mathbf{T}_{\mathbf{w_I}}\mathbf{v'_w})} \tag{3.7}$$

The model was made more efficient by approximating the full softmax with a hierarchical version which was first introduced by Morin and Bengio [77]. The hierarchical softmax uses a binary Huffman tree representation of the output layer with the words as its leaves and, for each node, represents the relative probabilities of its child nodes explicitly. This tree representation has a significant positive effect on the overall speed of the model. For more information on the hierarchical softmax and the continuous skip-gram model in general, we refer the reader to [71] and [73].

An interesting observation was made in [74]: the word embeddings that are implicitly learned by the network are surprisingly good at capturing

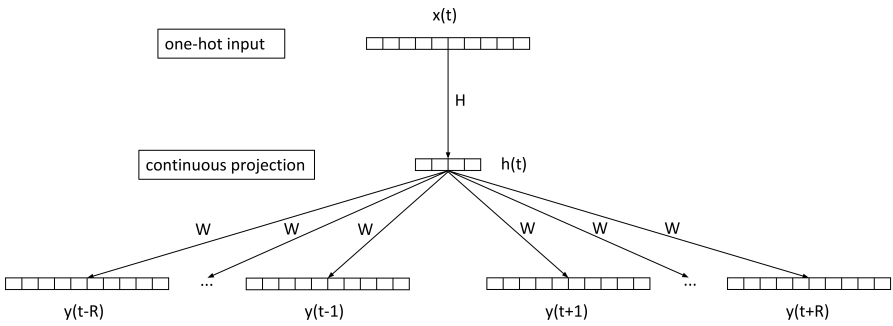

Figure 3.2: Continuous Skip-gram Model.

both semantic and syntactic regularities in language and each relationship is characterized by a relation-specific vector offset. This allows intuitive vector mathematics based on the offsets between words e.g. *King - Man + Woman* results in a vector that is very close to that of *Queen*. It is clear that these CSM word embedding vectors can be a valuable source of information to enrich existing language modeling techniques.

3.2.2 Semantic Language Model

For word embeddings to be incorporated into a language model, we need to have a representation at a higher level than just individual words. A naive notion of a history can be achieved by calculating the centroid of the previous K word vectors, but this makes the unreasonable assumptions that the meaning of a phrase is equal to the sum of its components and that all words in the history are equally important.

Mikolov et al. [73] showed that the first assumption can be overcome in part by detecting common phrases i.e. words that appear frequently together and infrequently in other contexts. This way they were able to replace the phrase *New York Times* by a single token while the phrase *this is* remained unchanged.

They did not however address the assumption of equal word importance. Function words like *the* and *of* are clearly less informative than content words, hence should be given less weight. This can be dealt with quite easily by weighting the embedding of each word in the history, using the same TF-IDF weights that were mentioned in Section 3.1.2. The history can then be represented as a weighted average of the word and/or phrase embeddings it contains.

Finally, one may wonder whether the vector addition of many words is still meaningful. It is likely that the context used in language models based on word embedding vectors should either be limited in length or processed in some hierarchical way where documents are recursively decomposed into smaller meaningful units.

In this work, we do not attempt phrase detection nor do we build hierarchical models, but instead we apply TF-IDF weighting to the word embeddings of the history to acquire a model that is similar to the LSA-based model presented in Section 3.1.2:

$$P_{CSM}(w_q|\tilde{d}_{q-1}) = \frac{[\cos(\mathbf{v_q}, \tilde{\mathbf{v}}_{\mathbf{q-1}}) - \min_{\mathbf{v}} \cos(\mathbf{v}, \tilde{\mathbf{v}}_{\mathbf{q-1}})]^\gamma}{\sum_{w_i \in \mathcal{V}} [\cos(\mathbf{v_i}, \tilde{\mathbf{v}}_{\mathbf{q-1}}) - \min_{\mathbf{v}} \cos(\mathbf{v}, \tilde{\mathbf{v}}_{\mathbf{q-1}})]^\gamma} \tag{3.8}$$

where $\mathbf{v_q}$ is the CSM word embedding of word w_q and $\tilde{\mathbf{v}}_{\mathbf{q-1}}$ is the weighted average of the word embeddings of the history $\tilde{d}_{q-1}$, with TF-IDF weights $\lambda(w)$:

$$\tilde{\mathbf{v}}_{\mathbf{q-1}} = \frac{1}{|\tilde{d}_{q-1}|} \sum_{w \in \tilde{d}_{q-1}} \lambda(w) \mathbf{v}_w \tag{3.9}$$

3.3 Experiments

In this section, we compare the combination of the aforementioned semantic language models with n-gram language models. Model fitness is measured as test set perplexity and compared with a 3-gram baseline model with modified Kneser-Ney smoothing. This form of smoothing has been shown to outperform the other well-known smoothing methods on a number of occasions and our case did not prove any different. We also trained 4-gram and 5-gram language models, but since the reduction in perplexity was minimal and since we are mostly interested in the reduction caused by adding word similarity information, we chose the 3-gram model to interpolate with the other models. We do however compare our final interpolated models to a 4-gram and 5-gram LM to show that our techniques can be more interesting than simply increasing the LM context.

All of the trained models are combined using linear interpolation. Although non-linear interpolation has been shown to achieve larger perplexity reductions [22], this better performance comes at the cost of higher complexity, since the LM scores need to be normalized for each evaluation. As our final goal is to apply these models in the context of automatic speech recognition, we prefer a fast combination technique. Note that whenever we report interpolation weights, we mean the weight attributed to the model currently under discussion and not the weight of the n-gram LM.

Our LM training data consists of a collection of normalized newspaper texts from the Flemish digital press database Mediargus. All of the models were trained on excerpts from the Southern Dutch newspaper De Standaard, which together contain 65M word tokens. The training set was not marked with document boundaries, but instead a document length of 30 sentences was assumed. Vocabulary selection was based on the most frequent words in this data set. Parameter optimization was done on a development set consisting of excerpts from the Southern Dutch newspaper De Morgen, which contain 100k word tokens. Finally, we validated all of our models on two test sets: excerpts of the Southern Dutch magazine Knack and Northern Dutch newspaper NRC, both of which were limited to 50k word tokens. In addition, we double-checked our conclusions by looking at similar length excerpts of two Southern Dutch

	Vocabulary size		
Data set	50k	100k	200k
De Morgen (dev)	4.84	2.93	1.81
Knack (test)	5.05	3.00	1.81
NRC (test)	6.69	4.38	3.01
Gazet van Antwerpen (test)	5.24	3.00	1.77
Het Belang van Limburg (test)	5.61	3.45	2.23

Table 3.1: OOV rates (in %) for the different data sets and vocabulary sizes in the experiments.

newspapers: Gazet van Antwerpen and Het Belang van Limburg. Words that are out of vocabulary (OOV) were mapped to an `<UNK>` token which was not included in the vocabulary. Table 3.1 shows the OOV rates on each of these data sets for the various vocabulary sizes that we experimented with.

All of the word similarity models were trained using the open source Python framework gensim [93], which contains intuitive and scalable implementations for many popular similarity models. The n-gram language models were trained with the SRILM toolkit [107].

The models using LSA and CSM are all preprocessed using TF-IDF. No phrase detection whatsoever was performed.

For each of the combined models, we performed an exhaustive grid search in their parameter space for vocabularies of size 50000, 100000 and 200000. Unless mentioned otherwise, we only show the results of a 100000 words vocabulary for the sake of clarity. We investigate what the optimal values are, which parameters are most influential and whether or not these parameters are robust by looking at their effect on perplexity. We also compare common parameters across models e.g. dimensionality and interpolation weight. Note that whenever a particular parameter is discussed, the remaining parameters are set to their optimal value with regards to the first parameter, thus avoiding any opaque results by averaging effects.

The experiments can be summarized as follows: first we explore cache models and investigate whether exponential decay is worth the extra computation. In Section 3.3.2, we compare the dimensionality of LSA and CSM. Next, in Section 3.3.3, we study the effect of the context window. Then, in Section 3.3.4, we compare each of the models with regards to their complexity. Finally, we review the best models, discuss their performance on two test sets and make some general conclusions on the applicability of Southern Dutch training data for Northern Dutch applications.

3.3.1 Cache Models: To Forget or not to Forget

Cache models have been shown to work better when attributing less weight to older words i.e. using a word influence decay [21]. In this section, we first investigate to what extent this is true for our data and which type of decay is the most promising. We also examine some general properties of our cache models.

Table 3.2 shows the perplexity of different interpolated n-gram+cache models on the development set as a function of the decay rate α and the size of the cache memory K. No decay is indicated by a value of $\alpha = 1$, all the other values represent exponential decay. As can be seen, the difference in performance between no decay and exponential decay is negligible. No decay performs better on somewhat smaller memories; most likely they attribute too much weight to older, less relevant words. Exponential decay is capable of capturing more distant information with a minimal perplexity of 198 for $0.992 \leq \alpha \leq 0.996$ and $1000 \leq K \leq 2000$, although the difference by increasing the size of the memory is minimal. Notice that the effect on perplexity decreases and sometimes even vanishes completely as the window gets larger, especially for low values of α. This is because the decay is so rapid that the influence of older words quickly reaches zero.

The optimal parameters of the cache models are more or less constant over different vocabulary sizes and are relatively robust with a maximal perplexity difference of around 20. The worst performance is observed for setups with a high cache interpolation weight and a very short window or low α. The optimal cache interpolation weight is 0.1 for most setups. The best setup is capable of reducing the perplexity by 24 over the 3-gram LM baseline on the development set.

We conclude that when choosing a setup for cache models, exponential decay is only marginally better than no decay. However, since the added complexity is minimal, the only reason not to use it is when parameter optimization is not an option. In the following experiments, we will therefore only refer to the optimal cache model with exponential decay.

3.3.2 On the Dimensionality of Word Similarity Models

In this section, we investigate how many features LSA and CSM need to store useful information about words. Less features needed means less storage needed which is a desirable property, but do more features always contain more information? What is the optimal dimensionality for each model, how do they

	α							
K	0.95	0.97	0.99	0.992	0.994	0.996	0.998	1
0 (3-gram)	222	222	222	222	222	222	222	222
50	214	213	213	213	213	213	213	213
250	209	204	200	200	200	201	201	201
500	209	204	199	199	199	199	200	201
1000	209	204	199	**198**	**198**	**198**	200	203
2000	209	204	199	**198**	**198**	**198**	200	207

Table 3.2: Perplexity results for interpolated n-gram+cache models with a vocabulary of 100000 words, as a function of decay rate α and cache memory size K, as measured on the De Morgen development set.

compare to each other and which model has the highest information-per-feature ratio i.e. which model is able to get the best results with the least amount of features?

Table 3.3 shows the results of different models using either LSA or CSM as word similarity component. Their performance is measured as perplexity on the development set as a function of the dimensionality of the vector space. It can be seen that the models using LSA already perform well at a dimensionality of 50, but very quickly reach saturation around 150 with a marginal perplexity reduction for a higher number of dimensions. The word embedding vectors also perform well at a dimensionality of 50, but they gain more by additional dimensions until a dimensionality of around 500. This causes them to outperform LSA by an increasing margin, albeit small, even at the highest number of dimensions.

The optimal parameters of the different models are, as was the case with the cache models, similar over different vocabulary sizes with, for LSA, a window of 200, LSA interpolation weight of 0.08, dimensionality of 1000 and $\gamma = 7$, and, for CSM, a window of 100, CSM interpolation weight of 0.07, dimensionality of 1000, $\gamma = 10$ and training context R of 240. The robustness of the CSM parameters is comparable to that of cache models with a maximal perplexity difference of around 25. The parameters of LSA are even more robust than those of cache models with a maximal perplexity difference of only around 10.

We conclude that CSM is capable of finding more interesting features in the data, yielding a small advantage over LSA in our development set. We believe that this difference will manifest itself more clearly as the size of the data increases.

	dimensionality					
	50	150	250	500	750	1000
3-gram	222	222	222	222	222	222
3-gram+LSA	211	205	203	202	201	201
3-gram+CSM	209	204	202	198	197	**195**

Table 3.3: Perplexity results for n-gram LMs, interpolated with LSA and CSM with a vocabulary of 100000 words, as a function of dimensionality, as measured on the De Morgen development set.

3.3.3 Windows: Learning from the Distant Past

Each of the models uses a window of the most recently observed words to evaluate which word is most likely to be observed next. N-gram LMs are forced to keep the size of their window very small, because they preserve the word order within the window which rapidly leads to sparsity problems with larger windows. Word similarity models are typically bag-of-words models i.e. they do not preserve the word order within the window and therefore do not suffer from this restriction. It is not clear however how much information resides in the very distant context words and whether a model is able to extract this information. In this section, we investigate what the optimal window sizes are for all three of the examined models and attempt to explain any differences therein. We discuss why certain models are not capable of extracting valuable information from distant context as well as others.

Table 3.4 shows the perplexity results of all the models on the development set, as a function of the size of the window. As was already discovered in Section 3.3.1, cache models with exponential decay are capable of retrieving valuable information from increasingly large contexts. The opposite is true for LSA and CSM, where we observe a rapid saturation and even degradation with increasing window size. We assume this is because, as opposed to cache models, these models capture real semantic information which is related to the ongoing topic. In natural language, and in particular in newspaper material such as our development set, these topics can change quite quickly. We believe this is also one of the reasons why cache models often outperform real similarity models, as they are able to attribute high probabilities to function words as well as content words.

LSA and CSM behave quite similarly with regards to context window size. Both saturate around 150 and then slowly degrade. Unexpectedly, CSM does not degrade more rapidly than LSA. As we mentioned in Section 3.2, we expected that averaging vectors for an increasing amount of words would yield a word

	window size							
	50	100	150	200	250	500	1000	2000
3-gram	222	222	222	222	222	222	222	222
3-gram+cache	213	206	203	201	200	199	198	198
3-gram+LSA	205	202	201	201	202	204	207	211
3-gram+CSM	196	**195**	**195**	196	197	201	206	211

Table 3.4: Perplexity results for n-gram LMs, interpolated with cache models, LSA and CSM with a vocabulary of 100000 words, as a function of window size, as measured on the De Morgen development set.

vector which could be described as having all meanings and no meaning at all. Our results show that this is not the case and, even though there is no document concept in these models, degradation behavior is quite similar to LSA. We believe that this is largely due to the application of TF-IDF preprocessing, which reduces the effect of most of the words in the window.

Finally, we observe that the largest perplexity reduction is achieved by CSM, albeit by a small margin. Given the high quality of the vectors, we believe that smarter compositional models for word embedding vectors such as [105] will lead to a more significant difference in performance, although probably at the cost of reduced efficiency.

We conclude that unlike cache models, TF-IDF preprocessed LSA and CSM thrive in smaller contexts of several hundreds of words and that CSM is capable of making the most out of the least number of words.

3.3.4 Efficiency

In this section, we sidetrack for a moment from perplexity evaluation and investigate a more practical concern which is the efficiency of each model. We do not attempt to theoretically analyze the asymptotic computational complexity of each of the models, but rather we compare the runtime needed for each of the models to evaluate the Knack test set. Each reported runtime is the result of averaging the runtimes of 10 otherwise identical experiments.

Table 3.5 shows the evaluation runtime for different models, as measured on an Intel Core i5-2400 3.10 GHz processor with 1 core only. We compared the times for n-gram models interpolated with a cache model with a window of 2000 words to LSA and CSM with a window of 150 words and 500 dimensions. The simplicity of the cache model is apparent as it needs less than 2 minutes for

	runtime
3-gram+cache	1m50
3-gram+LSA	2h06m
3-gram+CSM	1h03m

Table 3.5: Evaluation runtimes for n-gram LMs, interpolated with cache models, LSA and CSM with a vocabulary of 100000 words, as measured on the Knack test set, using a Intel Core i5-2400 processor with 1 core.

a complete evaluation of the test set. LSA takes on average 2h06 to complete the evaluation, whereas CSM is capable of finishing the job in 1h03, exactly half the runtime of LSA. We believe this is because Eq. (3.5) requires both the target word and the history to be projected to the LSA space, whereas Eq. (3.8) operates on the already available CSM embeddings.

It is clear that CSM is nowhere near as efficient as cache models, but it is twice as fast as LSA, which makes it an interesting candidate for integration into a speech recognizer.

3.3.5 Test Set Evaluation

In this section, we evaluate all of the created models on two different test sets: the Southern Dutch magazine Knack and the Northern Dutch newspaper NRC. We first compare the interpolated models to the 3-gram LM baseline to highlight the reduction in perplexity. We also report perplexities for 4-gram and 5-gram LMs (both with modified Kneser-Ney smoothing) to indicate that our techniques cannot be simply matched by increasing the context of an n-gram LM, at least not for the investigated data sets. Next, we show that the best two models (cache and CSM) are complementary and that a combined model yields the best results. All of the experiments are repeated for vocabularies of 50000, 100000 and 200000 to investigate any effects due to vocabulary size. Finally, we compare the results for both test sets in an attempt to find out to what extent models built with Southern Dutch material are suitable for Northern Dutch and double-check our conclusions on two additional Southern Dutch newspapers.

Tables 3.6, 3.7 and 3.8 show perplexity results for all of the mentioned models on both test sets, for vocabulary sizes of 50000, 100000 and 200000 words respectively. It can be seen that all of the interpolated models outperform the 3-gram LM baseline as well as 4-gram and 5-gram LMs. This is true regardless of the vocabulary size and shows that the reported perplexity reductions cannot be achieved by simply increasing the n-gram context. Zooming in on the

	Knack		NRC	
	PPL	Diff	PPL	Diff
3-gram	210		202	
4-gram	198	5.83%	189	6.22%
5-gram	198	6.02%	189	6.51%
3-gram+cache	188	10.68%	181	**10.49%**
3-gram+LSA	186	11.53%	185	8.17%
3-gram+CSM	184	**12.71%**	182	9.87%
3-gram+cache+CSM	176	16.42%	172	14.98%
4-gram+cache+CSM	166	21.00%	162	19.81%
5-gram+cache+CSM	166	21.02%	162	19.87%

Table 3.6: Perplexity results for all of the models with a vocabulary of 50000 words, as measured on the Knack and NRC test sets.

	Knack		NRC	
	PPL	Diff	PPL	Diff
3-gram	242		233	
4-gram	228	5.88%	219	6.20%
5-gram	227	6.11%	218	6.55%
3-gram+cache	213	11.81%	205	**11.87%**
3-gram+LSA	214	11.35%	213	8.45%
3-gram+CSM	210	**13.29%**	208	10.64%
3-gram+cache+CSM	199	17.69%	195	16.50%
4-gram+cache+CSM	188	22.23%	184	21.21%
5-gram+cache+CSM	188	22.28%	183	21.32%

Table 3.7: Perplexity results for all of the models with a vocabulary of 100000 words, as measured on the Knack and NRC test sets.

interpolated models, we notice that LSA consistently scores worst, except for Knack with a vocabulary of 50000 words, where it does better than the cache model. CSM is always the best model for Knack, although only by a small margin when using a vocabulary of 200000 words. Cache models have no competition whatsoever when evaluated on NRC. Finally, it is clear that cache models and CSM are complementary as the models that combine both cache and CSM outperform the others by a large degree with perplexity reductions of up to 23%.

The size of the vocabulary has the largest effect on the cache models, with performance going up with increasing vocabulary size. This is a logical result as more words can be stored in the cache memory and it is also visible in

	Knack		NRC	
	PPL	Diff	PPL	Diff
3-gram	267		261	
4-gram	251	5.93%	245	6.20%
5-gram	250	6.20%	244	6.56%
3-gram+cache	232	13.12%	227	**13.32%**
3-gram+LSA	238	11.05%	241	7.63%
3-gram+CSM	232	**13.19%**	235	10.04%
3-gram+cache+CSM	217	18.63%	216	17.36%
4-gram+cache+CSM	205	23.16%	204	22.04%
5-gram+cache+CSM	205	23.23%	203	22.14%

Table 3.8: Perplexity results for all of the models with a vocabulary of 200000 words, as measured on the Knack and NRC test sets.

the models combining both cache and CSM. LSA and CSM do not seem to be affected much by vocabulary size, although there is some evidence that performance drops somewhat when using too many words.

When we compare the results of Knack with those of NRC, it can be seen from any of the three tables, that the 3-gram LM baseline yields the lowest perplexity on the Northern Dutch newspaper NRC. As the training set also consisted of newspaper excerpts, this leads us to believe that n-gram LMs capture language information that is mostly related to the register of the training material. We assume that both the word choice and the syntactic structure of a newspaper are somewhat more formal than those of a magazine. The fact that Southern and Northern Dutch differ in word usage does not have any observable effect, perhaps due to the nature of the material. It is likely that newspapers, with their formal registers and broad target group, tend to choose more standard words, thus diminishing this language discrepancy.

If we look at the effect of the different models on the baseline, we notice that cache models have a similar effect on both test sets, which is not surprising as cache models are not trained and words are likely to be repeated in most if not all languages. We do however observe a large discrepancy in the semantic language models where both LSA and CSM show a larger perplexity reduction on the Southern Dutch magazine Knack. Although the reductions on the Northern Dutch newspaper NRC are still substantial, the discrepancy is a clear indication that there is in fact a difference between Northern and Southern Dutch with regards to language modeling. We believe this is mostly caused by the relatively higher OOV rate of NRC than that of Knack. That is, LSA and CSM can only take advantage of a word in the history if they built a

projection for this word during training. As this does not happen for OOV words, it explains why a higher OOV rate yields lower perplexity reductions. An alternative explanation lies in the co-occurrence of words. As both these models are based on the idea of co-occurrence, it might be the case that, for our data, words co-occur differently in Southern Dutch than they do in Northern Dutch. It is possible that this is due to a topic effect i.e. that certain topics occur more often in Southern Dutch texts, for example because they address a regional phenomenon. Finally, the discrepancy might also be explained by the fact that words can have a different meaning in Southern and Northern Dutch which yields different lexical relations.

Whatever the underlying cause, we conclude that Southern Dutch data can be used successfully as training material for the evaluation of Northern Dutch, especially when the intended models are n-grams and when there is a similarity in register. When the intended models concern word similarity, Southern Dutch texts can still prove valuable, but Northern Dutch data is more likely to be preferred.

To give further confidence to these conclusions we tested our models on two additional Southern Dutch newspapers: Gazet van Antwerpen and Het Belang van Limburg. Table 3.9 shows the perplexity results for these two test sets for a vocabulary size of 50000 words. It can be seen that the 3-gram LM baseline yields perplexities that are comparable to that of NRC. This strengthens our claim that n-gram LMs are hardly if at all influenced by language variety, but are in fact affected by style. On the other hand, when we look at the semantic language models, the perplexity reductions strongly resemble those of Knack, especially for CSM, reflecting the fact that they are all Southern Dutch publications.

	Gazet van Antwerpen		Het Belang van Limburg	
	PPL	Diff	PPL	Diff
3-gram	205		199	
4-gram	193	6.09%	185	7.46%
5-gram	192	6.25%	185	7.69%
3-gram+cache	181	11.88%	180	9.92%
3-gram+LSA	183	10.71%	182	9.18%
3-gram+CSM	181	**11.98%**	177	**11.29%**

Table 3.9: Perplexity results for all of the models with a vocabulary of 50000 words, as measured on the additional Southern Dutch newspapers.

3.4 ASR Integration

Although cache models are fast and powerful for text prediction, when applied to automatic speech recognition, they suffer from so-called *error propagation*. That is, contrary to text, in speech recognition the history is uncertain which means that incorrect word hypotheses may also be stored in the cache, thereby influencing the recognizer's future predictions in a negative way. Once an incorrect word has been given a high probability, the recognizer might make the same mistake over and over again. Moreover, the use of a cache model is limited when applied to time-synchronous decoding as keeping track of the many possible hypotheses over a long period of time quickly puts too much strain on the memory. A time-asychronous decoder therefore seems necessary to exploit the full potential of a cache model.

We believe that semantic models based on LSA and CSM suffer less from error propagation as their continuous word and history representations make them more robust with regards to syntactic errors. For example, if the recognizer incorrectly hypothesizes *cats* instead of *cat*, the model will probably not be affected, as their vector representations in the latent space are bound to be close. The models might even be robust with regards to minor semantic errors as long as the correct and incorrect word are semantically similar e.g. *airplane* and *airport*. And as long as most of the hypotheses are correct or in the correct domain, the history representation should also be in the correct domain, which reduces the probability of error propagation. Because of this robustness, it might also be possible to merge similar hypotheses, thereby shrinking the search space and making the use of a time-synchronous decoder more feasible.

Finally, the semantic model based on CSM has an additional advantage in that the word similarities can be precomputed and converted into probabilities. This further reduction in complexity makes the model more likely to be used in a first-pass decoding.

3.5 Conclusion

We have examined several combinations of n-gram language models with semantic language models including cache models, models based on LSA and our proposed models based on CSM, and made the following observations.

First, we did a thorough investigation of the model parameters and found that (1) cache models perform best when using exponential decay, although only marginally better than without decay; (2) CSM makes optimal use of larger

dimensions which yields it a small, but consistent perplexity reduction over LSA; (3) both CSM and LSA are optimal with short context windows whereas cache models typically prefer longer windows.

Next, we compared the addition of each model to a 3-gram LM with modified Kneser-Ney and found that the continuous skip-gram model is a viable alternative for LSA in the context of interpolated language models. Not only does it achieve consistently lower perplexities, it is also twice as fast and combines well with cache models. Our final model which uses an interpolation of a 3-gram LM, a cache model and a continuous skip-gram model was able to reduce the perplexity of a 3-gram LM by as much as 18.63%. This is about three times the reduction achieved with a 5-gram LM.

Finally, we compared the effect of LMs trained with Southern Dutch material on both a Southern Dutch and a Northern Dutch test set. We conclude that there is no outspoken difference in performance for n-gram LMs, which suggests that they are mostly influenced by the register of the training material. When it comes to word similarity on the other hand, material of the same language (variety) seems to be desirable. These conclusions were further confirmed by looking at two additional Southern Dutch newspapers.

Chapter 4

Sparse Non-negative Matrix Language Modeling

In the previous chapter, we proposed the linear interpolation of n-gram and semantic models in an attempt to have the best of both worlds: n-grams capture local language phenomena whereas semantic models capture more global phenomena. Although this proved to be a successful strategy, linear interpolation is a suboptimal combination scheme: it always attributes the same weight to each model without taking into account the word sequence that is currently under investigation.

In this chapter, we also want to model more global phenomena, but rather than combining two models, we propose the Sparse Non-negative Matrix (SNM) language model that can efficiently incorporate arbitrary features. This joint modeling should enable the features to be weighted by their relative importance for the current word sequence. We demonstrate that the model can leverage long-distance context by training models with variable-length n-gram features and skip-gram features to incorporate long-distance context.

The chapter is organized as follows: Section 4.1 gives the motivation for Sparse Non-negative Matrix language models which are described in Section 4.2. We then present a complexity analysis in Section 4.3 and experimental results on two English corpora in Sections 4.4 and 4.5. Finally, in Section 4.7, we present some thoughts on how SNM models could be integrated into the first pass of a speech recognizer. We end with a conclusion in Section 4.8.

4.1 Motivation

In Chapter 1, we mentioned that more advanced techniques such as neural networks and Maximum Entropy that attempt to model long-distance phenomena have the disadvantage of being computationally complex. In this section, we will briefly remind the reader why this is the case and motivate the use of skip-grams as features for our more efficient alternative proposal.

4.1.1 Log-linear Models

Neural networks and Maximum Entropy (ME) are related in the sense that for both models the probability of word w_k given the equivalence classifier of the history $\Phi(w_1^{k-1})$ takes the following log-linear form:

$$P(w_k|\Phi(w_1^{k-1})) = \frac{\exp(\hat{y}_{w_k})}{\sum_{t' \in \mathcal{V}} \exp(\hat{y}_{t'})} \tag{4.1}$$

where $\hat{\mathbf{y}}$ is the vector of unnormalized log-probabilities and each $\hat{y}_{t'}$ represents the unnormalized log-probability for a potential target word t'. How $\hat{\mathbf{y}}$ is computed, depends on the model in question. For a ME model with features $\mathcal{F}$, $\hat{\mathbf{y}}$ can be represented as follows:

$$\hat{\mathbf{y}} = \mathbf{x}^{\mathbf{T}}\mathbf{M} \tag{4.2}$$

where $\mathbf{x}$ is a feature activation vector of size $|\mathcal{F}|$ and $\mathbf{M}$ is a $|\mathcal{F}| \times |\mathcal{V}|$ feature weight matrix. The unnormalized log-probabilities of neural networks on the other hand are computed as follows:

$$\hat{\mathbf{y}} = g(\mathbf{x}^{\mathbf{T}}\mathbf{H})\mathbf{W} \tag{4.3}$$

where $g(\cdot)$ is the activation function of the hidden layer (typically a tanh or sigmoid) and $\mathbf{W}$ and $\mathbf{H}$ are weight matrices for the output and hidden layer respectively. Feed-forward and recurrent neural networks differ only in their input vectors $\mathbf{x}$: in a feed-forward neural network, $\mathbf{x}$ is a concatenation of the input features whereas in a recurrent neural network, $\mathbf{x}$ is a concatenation of the input word and the previous hidden state. Because of their shared log-linearity, training and evaluating these models becomes computationally complex.

Although log-linear models have been shown to perform better than linear models [60], their performance is also hampered by their complexity and we will show in this chapter that a linear model can in fact compete with the state of the art when trained with variable-length n-gram and skip-gram features combined.

4.1.2 Skip-grams

The type of long-distance features that we incorporate into our SNMLMs are skip-grams [50,78,96], which can effectively capture dependencies across longer contexts. We are not the first to highlight this effectiveness; previous such results were reported in [104]. Recently, [89] also showed that a back-off generalization using single skips yields significant perplexity reductions. We note though that our SNMLMs are trained by mixing single as well as longer skips, combining both in one model. More fundamentally, the SNM model parameterization and method of estimation are completely original, as far as we know.

We characterize a skip-gram feature extracted from the context w_1^{k-1} by the tuple (r, s, a) where:

- r denotes the number of remote context words
- s denotes the number of skipped words
- a denotes the number of adjacent context words

relative to the target word w_k being predicted. The window size of a feature extractor then corresponds to $r + s + a$. For example, in the sentence `<S> The quick brown fox jumps over the lazy dog </S>` a $(1, 2, 3)$ skip-gram feature for the target word `dog` is:

```
[brown skip-2 over the lazy]
```

We configure the skip-gram feature extractor to produce all features $\mathcal{F}$, defined by the equivalence class $\Phi(w_1^{k-1})$, that meet constraints on the minimum and maximum values for:

- the number of context words $r + a$
- the number of remote words r
- the number of adjacent words a
- the skip length s

We also allow the option of not including the exact value of s in the feature representation; this may help with smoothing by sharing counts for various skip features. The resulting tied skip-gram features look like this:

```
[curiosity skip-* the cat]
```

where the $*$ indicates that any number of words can be skipped.

In order to build a good probability estimate for the target word w_k in a context w_1^{k-1} we need a way of combining an arbitrary number of skip-gram features, which do not fall into a simple hierarchy like regular n-gram features. The standard way to combine such predictors is ME, but as we saw in Section 4.1.1, it is computationally hard. The proposed SNM estimation on the other hand is capable of combining such predictors in a way that is computationally easy, scales up gracefully to large amounts of data and as it turns out is also very effective from a modeling point of view.

4.2 Sparse Non-negative Matrix Estimation

4.2.1 Linear Model

Contrary to neural networks and ME, SNM language models do not estimate $P(w_k|\Phi(w_1^{k-1}))$ in a log-linear fashion, but are in fact linear models:

$$P(w_k|\Phi(w_1^{k-1})) = \frac{\hat{y}_{w_k}}{\sum_{t' \in \mathcal{V}} \hat{y}_{t'}} \tag{4.4}$$

where $\hat{\mathbf{y}}$ is defined as in Eq. (4.2).

Like ME however, SNM uses features $\mathcal{F}$ that are predefined and arbitrary, e.g. n-grams, skip-grams, bags of words, syntactic features, ... The features are extracted from the left context of w_k and stored in a feature activation vector $\mathbf{x} = \Phi(w_1^{k-1})$, which is binary-valued, i.e. x_f represents the presence or absence of the feature with index f.

In what follows, we represent the target word w_k by a vector $\mathbf{y}$, which is a one-hot encoding of the vocabulary $\mathcal{V}$: $y_t = 1$ for $t = w_k$, $y_t = 0$ otherwise. To further simplify notation, we will not make the distinction between a feature/target and its index, but rather denote both of them by f and t, respectively.

The $\hat{y}_{t'}$ in SNM are computed in the same way as ME, using Eq. (4.2), where $\mathbf{M}$ is a $|\mathcal{F}| \times |\mathcal{V}|$ feature weight matrix, which is sparse and non-negative. M_{ft} is indexed by feature f and target t and denotes the influence of feature f in the prediction of t. Plugging Eq. (4.2) into Eq. (4.4), we can derive the complete

form of the conditional distribution $P(\mathbf{y}|\mathbf{x}) = P(w_k|\Phi(w_1^{k-1}))$ in SNMLMs:

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= \frac{(\mathbf{x}^\mathbf{T}\mathbf{M})_{w_k}}{\sum_{t' \in \mathcal{V}} (\mathbf{x}^\mathbf{T}\mathbf{M})_{t'}} \\ &= \frac{\sum_{f' \in \mathcal{F}} x_{f'} M_{f'w_k}}{\sum_{t' \in \mathcal{V}} \sum_{f' \in \mathcal{F}} x_{f'} M_{f't'}} \\ &= \frac{\sum_{f' \in \mathcal{F}} x_{f'} M_{f'w_k}}{\sum_{f' \in \mathcal{F}} x_{f'} \sum_{t' \in \mathcal{V}} M_{f't'}} \end{aligned} \tag{4.5}$$

As required by the denominator in Eq. (4.5), this computation also involves summing over all the present features for the entire vocabulary. However, because of the linearity of the model, we can precompute the row sums $\sum_{t' \in \mathcal{V}} M_{f't'}$ for each f' and store them together with the model. This means that the evaluation can be done very efficiently, since the remaining summation involves a limited number of terms: even though the amount of features $|\mathcal{F}|$ gathered over the entire training data is potentially huge, the amount of active, non-zero features for a given $\mathbf{x}$ is small. For example, for SNM models using variable-length n-gram features, the maximum number of active features is n; in our experiments with a large variety of skip-grams, it was around 100. Moreover, since most features only co-occur with a handful of target words in the training data, the rows of $\mathbf{M}$ are sparse, which means that precomputing the row sums for each feature can also be done efficiently.

Notice that this precomputation is not possible for the log-linear ME which is otherwise similar, because the sum over all features does not distribute outside the sum over all targets in the denominator:

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp(\sum_{f' \in \mathcal{F}} x_{f'} M_{f'w_k})}{\sum_{t' \in \mathcal{V}} \exp(\sum_{f' \in \mathcal{F}} x_{f'} M_{f't'})} \tag{4.6}$$

This is a huge difference and essentially makes SNM a more efficient model at runtime.

4.2.2 Adjustment Function and Meta-features

We let the entries of $\mathbf{M}$ be a slightly modified or *adjusted* version of the relative frequencies:

$$M_{ft} = e^{A(f,t)} \frac{C_{ft}}{C_{f*}} \tag{4.7}$$

where $A(f,t)$ is a real-valued function, dubbed the *adjustment function* (to be defined below), and $\mathbf{C}$ is a feature-target count matrix, computed over the entire training corpus $\mathcal{T}$. C_{ft} denotes the co-occurrence count of feature f and target t, whereas C_{f*} denotes the total occurrence count of feature f, summed over all targets t'.

An unadjusted SNM model, where $A(f,t) = 0$, is a linear mixture of simple feature models $P(t|f)$ with uniform mixture weights. The adjustment function enables the models to be weighted by the relative importance of each input feature and, because it is also parameterized by t, takes into account the current target. Notice that, although the non-linear adjustment function may seem to nullify the effect of the linear relative frequencies C_{ft}/C_{f*} in Eq. (4.7), keeping the relative frequencies enables fast initialization and cheap tuning. That is, the model can be efficiently initialized by gathering counts from the entire training data, after which the computationally more demanding adjustment function tuning can be done on a smaller part of the data (see also Section 4.2.4).

The adjustment function is computed by a linear model on binary *meta-features* [68]:

$$A(f,t) = \boldsymbol{\theta} \cdot \mathbf{h}(f,t) \tag{4.8}$$

where $\mathbf{h}(f,t)$ is the meta-feature vector extracted from the feature-target pair (f,t) and $\boldsymbol{\theta}$ is the vector containing the corresponding meta-feature weights. The idea behind meta-features is that features often have certain properties that have a potential relationship to the prediction problem. Estimating weights on the meta-feature level rather than the input feature level therefore enables features that share these properties to also share weights which improves generalization. We illustrate this by an example.

Given the word sequence `the quick brown fox`, we extract the following elementary meta-features from the 4-gram feature `the quick brown` and the target `fox`:

- feature identity = `[the quick brown]`
- feature type = 4-gram
- feature count = $C_{\texttt{[the quick brown]}*}$
- target identity = `fox`
- feature-target count = $C_{\texttt{[the quick brown] fox}}$

We also allow conjunctions of meta-features to form more complex meta-features. For example, by joining the feature identity and the target identity,

we get the feature-target identity which can be used to learn that `[the quick brown]` is a good predictor of `fox`. Alternatively, by joining the feature type with the target identity, the model could learn that 4-gram features do not offer much added value over 2-gram features when it comes to predicting `Francisco`. Operating at the meta-feature level also automatically introduces a form of smoothing as weights of infrequent features – which are often unreliable – are compensated by the shared weights of the more frequent elementary meta-features. For more information about which conjunctions are allowed, we refer to the pseudocode in Appendix A. Figure 4.1 shows a schematic overview of the model with only elementary meta-features for the sake of simplicity.

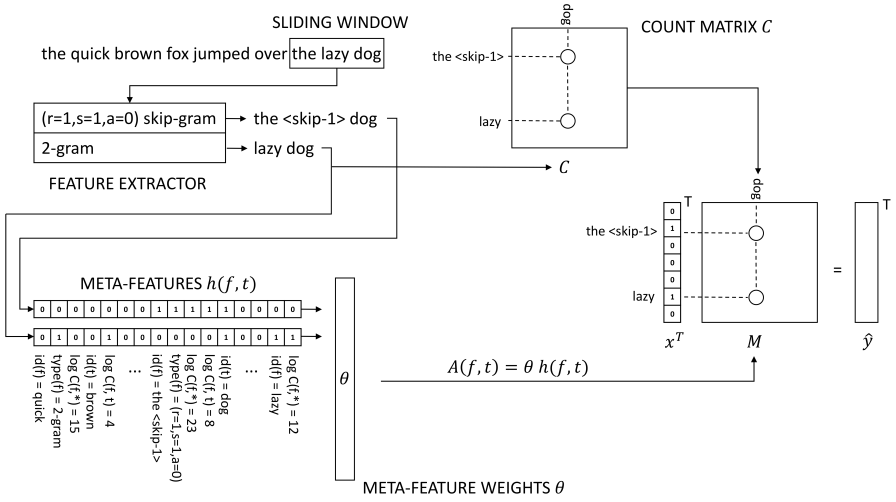

Figure 4.1: Sparse Non-negative Matrix Language Model.

We apply additional smoothing to all of the count meta-features: since count meta-features of the same order of magnitude carry similar information, we group them so they can also share weights. We do this by bucketing the count meta-features according to their (floored) $\log_2$ value. As this effectively puts the lowest count values, of which there are many, into a different bucket, we optionally introduce a second (ceilinged) bucket to assure smoother transitions. Both buckets are then weighted according to the $\log_2$ fraction lost by the corresponding rounding operation. Pseudocode for meta-feature extraction and count bucketing is presented in Appendix A.

To control memory usage, we employ a feature hashing technique [41, 65] where we store the meta-feature weights in a flat hash table of predefined size. Strings are fingerprinted (converted into a byte array, then hashed), counts are

hashed, and the resulting integer is mapped to an index by taking its value modulo the predefined $size(\boldsymbol{\theta})$. We do not prevent collisions, which has the potentially undesirable effect of tying together the weights of different meta-features. However, as was previously observed by [72], when this happens the most frequent meta-feature will dominate the final value after training, which essentially boils down to a form of pruning. Because of this, the model performance does not strongly depend on the size of the hash table. Note that we only apply hashing to the meta-feature weights: the adjusted and raw relative frequencies are stored as SSTables (Sorted String Table).

4.2.3 Model Estimation

Although it is in principle possible to use regularized maximum likelihood to estimate the parameters of the model, a gradient-based approach would end up with parameter updates involving the gradient of the log of Eq. (4.5) which works out to:

$$\frac{\partial \log P(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f M_{ft} \left(\frac{y_t}{\hat{y}_{w_k}} - \frac{1}{\sum_{t' \in \mathcal{V}} \hat{y}_{t'}} \right) \tag{4.9}$$

For the complete derivation, see Appendix B. The problem with this gradient is that we need to sum over the entire vocabulary $\mathcal{V}$ in the denominator. In Eq. (4.5) we could get away with this by precomputing the row sums, but here the sums change after each update. Instead, we were inspired by [115] and chose to use an independent binary predictor for each word in the vocabulary during estimation. Our approach however differs from [115] in that we do not use $|\mathcal{V}|$ Bernoulli distributed variables, but $|\mathcal{V}|$ Poisson distributed variables[1], using the fact that for a large number of trials a Bernoulli distribution with small p is well approximated by a Poisson distribution with small λ.

If we consider each $y_{t'}$ in $\mathbf{y}$ to be Poisson distributed with parameter $\hat{y}_{t'}$, the conditional probability $P_{\text{Pois}}(\mathbf{y}|\mathbf{x})$ is given by:

$$P_{\text{Pois}}(\mathbf{y}|\mathbf{x}) = \prod_{t' \in \mathcal{V}} \frac{\hat{y}_{t'}^{y_{t'}} e^{-\hat{y}_{t'}}}{y_{t'}!} = \prod_{t' \in \mathcal{V}} \hat{y}_{t'}^{y_{t'}} e^{-\hat{y}_{t'}} \tag{4.10}$$

where we dropped the denominator because $\mathbf{y}$ contains only binary values. The gradient of the log-probability works out to:

$$\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f M_{ft} \left(\frac{y_t}{\hat{y}_{w_k}} - 1 \right) \tag{4.11}$$

[1] We chose Poisson so we could apply the model to tasks with outputs $y_t > 1$.

For the complete derivation, see Appendix C.

The parameters $\boldsymbol{\theta}$ of the adjustment function are learned by maximizing the Poisson log-probability, using stochastic gradient ascent. That is, for each feature-target pair (f, t) we compute the gradient in Eq. (4.11) and propagate it to the meta-feature weights θ_k by multiplying it with $\partial A(f,t)/\partial\theta_k = h_k$. Each weight θ_k is then updated using the propagated gradient, weighted by a learning rate η:

$$\theta_k \leftarrow \theta_k + \eta \frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial \theta_k} \tag{4.12}$$

Rather than using a single fixed learning rate η, we employ the adaptive learning rate method AdaGrad [37] which keeps track of the per-weight sum of squared gradients to decay the learning rate $\eta_{k,N}$ for weight θ_k at time N:

$$\eta_{k,N} = \frac{\gamma}{\sqrt{\Delta_0 + \sum_{n=1}^{N} (\partial\theta_k)_n^2}} \tag{4.13}$$

where γ is a constant scaling factor for all learning rates, Δ_0 is an initial accumulator constant and $(\partial\theta_k)_n$ is a short-hand notation for the n^{th} gradient with respect to θ_k. Basing the learning rate on historical information tempers the effect of frequently occurring features which keeps the weights small and as such acts as a form of regularization.

4.2.4 Optimization and Leave-one-out Training

Each feature-target pair (f, t) constitutes a training example where examples with $y_t = 0$ are called negative and examples with $y_t = 1$ are called positive. Using the short-hand notations $T = |\mathcal{T}|$, $F = |\mathcal{F}|$ and $V = |\mathcal{V}|$, this means that the training data consists of approximately $TF(V-1)$ negative and only TF positive training examples. If we examine the two terms of Eq. (4.11) separately, we see that the first term $x_f M_{ft} \frac{y_t}{\hat{y}_{w_k}}$ depends on y_t which means it becomes zero for all the negative training examples. The second term $-x_f M_{ft}$ however does not depend on y_t and therefore never becomes zero. This also means that the total gradient is never zero and because of this, the vast amount of updates required for the negative examples makes the update algorithm computationally too expensive.

To speed up the algorithm we use a heuristic that allows us to express the second term as a function of y_t, essentially redistributing the updates for the numerous negative examples to the fewer positive training examples. Appendix D shows that for batch training this has the same effect if run over the entire corpus.

We note that for online training this is not strictly correct, since M_{ft} changes after each update. Nonetheless, we found this to yield good results as well as seriously reducing the computational cost. After applying the redistribution, the online gradient that is applied to each training example becomes:

$$\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f y_t M_{ft} \left(\frac{1}{\hat{y}_{w_k}} - \frac{C_{f*}}{C_{ft}} \right) \tag{4.14}$$

which is non-zero only for positive training examples, hence making training independent of the size of the vocabulary.

One practical way to further prevent overfitting and adapt the model to a specific task is to use held-out data, i.e. compute the count matrix $\mathbf{C}$ on the training data and estimate the parameters $\boldsymbol{\theta}$ on the held-out data. Unfortunately, since the aggregated gradients in Eq. (4.14) tie the updates to the counts C_{f*} and C_{ft} in the training data, they can't differentiate between held-out and training data, which means that the meta-feature weights can't be tuned specifically to the held-out data. Experiments in which we tried to use the held-out counts instead did not yield good results, presumably because the redistribution heuristic no longer works.

Rather than adding a regularizer on the meta-feature weights, we instead opted for leave-one-out training. With the notation $A(f,t,C_{f*},C_{ft})$ reflecting the dependence of the adjustment function on feature and feature-target counts, the gradient under leave-one-out training becomes:

$$x_f y_t \left((\frac{1}{\hat{y}^+_{w_k}} - 1) M^+_{ft} - \frac{C_{f*} - C_{ft}}{C_{ft}} M^-_{ft} \right) \tag{4.15}$$

where M^-_{ft}, M^+_{ft} and $\hat{y}^+_{w_k}$ are defined as follows:

$$M^-_{ft} = e^{A(f,t,C_{f*}-1,C_{ft})} \frac{C_{ft}}{C_{f*}-1}$$

$$M^+_{ft} = e^{A(f,t,C_{f*}-1,C_{ft}-1)} \frac{C_{ft}-1}{C_{f*}-1}$$

$$\hat{y}^+_{w_k} = (\mathbf{x}^T \mathbf{M}^+)_{w_k}$$

The full derivation can be found in Appendix E. We note that in practice, it often suffices to use only a subset of the training examples for leave-one-out training, which has the additional advantage of speeding up training even further.

4.3 Complexity Analysis

Besides their excellent results, RNNs have also been shown to scale well with large amounts of data with regards to memory and accuracy [112]. Compared to n-gram models which grow huge very quickly with only modest improvements, RNNs take up but a fraction of the memory and exhibit a near linear reduction in log perplexity with log number of training words. Moreover, a larger hidden layer can yield more improvements, whereas n-gram models quickly suffer from data sparsity. The problem with RNNs however is that they are computationally complex which makes training and evaluation slow. Training a standard Elman network [39] with hidden layer of size H on a corpus of size T with vocabulary of size V involves updating $1 \times H$ input-to-hidden weights, $H \times H$ hidden-to-hidden weights and $H \times V$ hidden-to-output weights for each training example. Its computational complexity is therefore:

$$IT(H + H^2 + HV) \tag{4.16}$$

where I indicates the number of iterations. Several attempts have been made to reduce training time, focusing mostly on reducing the large factors T or V:

- vocabulary shortlisting [101]
- subsampling [102, 115]
- class-based [44, 72, 77]
- noise-contrastive estimation [19, 45]

However, these techniques either come with a serious performance degradation [67] or do not sufficiently speed up training. The class-based implementation for example, which first estimates a probability distribution over C classes and then estimates a variable amount V_C of words in a class, still has a computational complexity of:

$$IT(H + H^2 + HC + CV_C) \tag{4.17}$$

Because $C \ll V$, this is a significant reduction in complexity, but unfortunately the dominant term ITH^2 is still large.

As was shown in [72], a Maximum Entropy model can be regarded as a neural network with direct connections for the features, i.e. it has no hidden layers. The model uses the same softmax activation at its output and its complexity therefore also depends on the size of the vocabulary:

$$IT(F_+V) \tag{4.18}$$

where $F_+ \ll F$ denotes the number of (active) features. To achieve state-of-the-art results this model is often combined with an RNN, which yields a total complexity of:

$$IT(H + H^2 + HV + F_+V) \tag{4.19}$$

The computational complexity for training SNM models on the other hand is independent of V:

$$TF_+ + IT'F_+\Theta_+ \tag{4.20}$$

where Θ_+ is the number of meta-features for each of the F_+ input features. The first term is related to counting features and feature-target pairs and the second term to training the adjustment model on a subset T' of the training data. If we compare an SNMLM with typical values of $F_+ \approx 100$ and $\Theta_+ < 40$, to the RNNLM configurations with $H = 1024$ in [14] and [112], we find that training comes at a reduced complexity of at least two orders of magnitude.

An even more striking difference in complexity can be seen at test time. Whereas the complexity of a class-based RNN for a single test step is proportional to $H + H^2 + HC + CV_C$, testing SNMLMs is linear in F_+, because of the reasons outlined in Section 4.2.1.

4.4 Experiment 1: 1B Word Benchmark

Our first experimental setup used the One Billion Word Benchmark[2] made available by [14]. It consists of an English training and test set of about 0.8 billion and 159658 tokens, respectively. The vocabulary contains 793471 words and was constructed by discarding all words with count below 3. OOV words are mapped to an `<UNK>` token which is also part of the vocabulary. The OOV rate of the test set is 0.28%. Sentence order is randomized.

All of the described SNM models are initialized with meta-feature weights $\theta_k = 0$ which are updated using AdaGrad with accumulator $\Delta_0 = 1$ and scaling factor $\gamma = 0.02$ over a single epoch of 30M training examples. The hash table for the meta-features was limited to 200M entries as increasing it yielded no significant improvements.

4.4.1 N-gram Experiments

In the first set of experiments, we used all variable-length n-gram features that appeared at least once in the training data up to a given length. This

[2] `http://www.statmt.org/lm-benchmark`

yields at most n active features: one for each m-gram of length $0 \leq m < n$ where $m = 0$ corresponds to an *empty feature* which is always present and produces the unigram distribution. The number of features is smaller than n when the context is shorter than $n - 1$ words (near sentence boundaries) and during evaluation where an n-gram that did not occur in the training data is discarded.

When trained using these features, SNMLMs come very close to the performance of n-gram models with interpolated Kneser-Ney (KN) smoothing [61], where no count cut-off was applied and the discount does not change with the order of the model. Table 4.1 shows that Katz smoothing [56] performs considerably worse than both SNM and KN. KN and SNM are not very complementary as linear interpolation with weights optimized on the test data only yields an additional perplexity reduction of about 1%. The difference between KN and SNM becomes smaller when we increase the size of the context, going from 5% for 5-grams to 3% for 8-grams, which indicates that SNMLMs might be better suited to a large number of features.

	n-gram order			
Model	5	6	7	8
KN	**67.6**	**64.3**	**63.2**	**62.9**
Katz	79.9	80.5	82.2	83.5
SNM (proposed)	70.8	67.0	65.4	64.8
KN+SNM	66.5	63.0	61.7	61.4

Table 4.1: Perplexity results on the 1B Word Benchmark for Kneser-Ney (KN), Katz and SNM n-gram models of different order.

4.4.2 Integrating Skip-gram Features

To incorporate skip-gram features, we can either build a 'pure' skip-gram SNMLM that contains no regular n-gram features (except for unigrams) and interpolate this model with KN, or we can build a single SNMLM that has both the regular n-gram features and the skip-gram features. We compared the two approaches by choosing skip-gram features that can be considered the skip-equivalent of 5-grams, i.e. they contain at most 4 context words. In particular, we configured the following feature extractors:

- $1 \leq r \leq 3$; $1 \leq s \leq 3$; $1 \leq r + a \leq 4$
- $1 \leq r \leq 2$; $s \geq 4$ (tied); $1 \leq r + a \leq 4$

We then built a model that uses both these features and regular 5-grams (SNM5-skip), as well as one that only uses the skip-gram features (SNM5-skip (no n-grams)). In addition, both models were interpolated with a KN 5-gram model (KN5).

As can be seen from Table 4.2, it is better to incorporate all features into one single SNM model than to interpolate with a KN 5-gram model (KN5). This is not surprising as linear interpolation uses a fixed weight for the evaluation of every word sequence, whereas the SNM model applies a variable weight that is dependent both on the context and the target word. Finally, interpolating the all-in-one SNM5-skip with KN5 yields almost no additional gain.

Model	PPL
SNM5-skip (no n-grams)	69.8
+ n-gram features = SNM5-skip	**54.2**
+ KN5 (interpolation)	56.5
SNM5-skip + KN5 (interpolation)	**53.6**

Table 4.2: Perplexity (PPL) results comparing two ways of adding n-grams to a 'pure' skip-gram SNM model (no n-grams): joint modeling (SNM5-skip) and linear interpolation with KN5.

4.4.3 Skip-gram Experiments

The best SNMLM results so far (SNM10-skip) were achieved using 10-grams, together with skip-grams defined by the following feature extractors:

- $s = 1$; $1 \leq r + a \leq 5$
- $r = 1$; $1 \leq s \leq 10$ (tied); $1 \leq r + a \leq 4$

This mixture of rich (large context) short-distance and shallow long-distance features enables the model to achieve state-of-the-art results. Table 4.3 compares its perplexity to KN5 as well as to the following language models:

- Stupid Back-Off LM (SBO) [9]
- Hierarchical Softmax Maximum Entropy LM (HSME) [44, 77]
- Recurrent Neural Network LM with Maximum Entropy (RNNME) [70]

Describing these models however is beyond the scope of this work. Instead, we refer the reader to [14] for a detailed description. The table also lists the number of model parameters, which in the case of SNMLMs consist of the non-zero entries and precomputed row sums of $\mathbf{M}$.

Model	Params	PPL
KN5	1.76 B	67.6
SNM5 (proposed)	1.74 B	70.8
SBO	1.13 B	87.9
HSME	6 B	101.3
SNM5-skip (proposed)	62 B	54.2
SNM10-skip (proposed)	33 B	**52.9**
RNNME-256	20 B	58.2
RNNME-512	20 B	54.6
RNNME-1024	20 B	**51.3**
SNM10-skip + RNNME-1024		**41.3**
KN5 + SBO + RNNME-512 + RNNME-1024		43.8
ALL		**41.0**

Table 4.3: Number of parameters and perplexity (PPL) results on the 1B Word Benchmark for the proposed models, compared to the models in [14].

When we compare the perplexity of SNM10-skip with the state-of-the-art RNNLM with 1024 hidden neurons (RNNME-1024), the difference is only 3%. Moreover, the small advantage that the RNNLM has, comes at the cost of increased training and evaluation complexity. Interestingly, when we interpolate the two models, we have an additional gain of 20%, which shows that SNM10-skip and RNNME-1024 are also complementary. At the time of publication, the resulting perplexity of 41.3 is already the best ever reported on this corpus, beating the optimized combination of several models, reported in [14] by 6%. Finally, interpolation over all models shows that the contribution of other models as well as the additional perplexity reduction of 0.3 is negligible.

4.4.4 Runtime Experiments

In this section, we present actual runtimes to give some idea of how the theoretical complexity analysis of Section 4.3 translates to a practical application. More specifically, we compare the training runtime (in machine hours) of the best SNM model to the best RNN and n-gram models:

- KN5: 28 machine hours
- SNM5: 115 machine hours
- SNM10-skip: 487 machine hours
- RNNME-1024: 5760 machine hours

As these models were trained using different architectures (number of CPUs, type of distributed computing, etc.), a runtime comparison is inherently hard and we would therefore like to stress that these numbers should be taken with a grain of salt. However, based on the order of magnitude we can clearly conclude that SNM's reduced training complexity shown in Section 4.3 translates to a substantial reduction in training time compared to RNNs. Moreover, the large difference between KN5 and SNM5 suggests that our vanilla implementation can be further improved to achieve even larger speed-ups.

4.5 Experiment 2: 44M Word Corpus

In addition to the experiments on the One Billion Word Benchmark, we also conducted experiments on a small subset of the LDC English Gigaword corpus. This has the advantage that the experiments are more easily reproducible and, since this corpus preserves the original sentence order, it also allows us to investigate SNM's capabilities of modeling phenomena that cross sentence boundaries.

The corpus is the one used in [108], which we acquired with the help of the authors and is now available online[3,4]. It consists of a training set of 44M tokens, a check set of 1.7M tokens and a test set of 13.7M tokens. The vocabulary contains 56k words which corresponds to an OOV rate of 0.89% and 1.98% for the check and test set, respectively. OOV words are mapped to an `<UNK>` token. The large difference in OOV rate between the check and test set is explained by the fact that the training data and check data are from the same source (Agence France-Presse), whereas the test data is drawn from CNA (Central News Agency of Taiwan) which seems to be out of domain relative to the training data. This discrepancy also shows in the perplexity results, presented in Table 4.4.

All of the described SNM models are initialized with meta-feature weights $\theta_k = 0$ which are updated using AdaGrad with accumulator $\Delta_0 = 1$ and scaling factor $\gamma = 0.02$ over a single epoch of 10M training examples. The hash table for the meta-features was limited to 10M entries as increasing it yielded no significant improvements.

With regards to n-gram modeling, the results are analogous to the 1B word experiment: SNM5 is close to KN5; both outperform Katz5 by a large margin. This is the case for the check set and the test set.

[3] `http://www.esat.kuleuven.be/psi/spraak/downloads/`

[4] In order to comply with the LDC license, the data was encrypted using a key derived from the original data.

	PPL	
Model	check	test
KN5	104.7	229.0
Katz5	124.1	292.6
SNM5 (proposed)	108.3	232.3
SLM	-	279
n-gram/SLM	-	243.0
n-gram/PLSA	-	196.0
n-gram/SLM/PLSA	-	176.0
SNM5-skip (proposed)	89.5	198.4
SNM10-skip (proposed)	87.5	195.3
SNM5-skip-`</S>` (proposed)	79.5	176.0
SNM10-skip-`</S>` (proposed)	78.4	174.0
RNNME-512	70.8	136.7
RNNME-1024	**68.0**	**133.3**

Table 4.4: Perplexity (PPL) results on the 44M corpus. On the small check set, SNM outperforms a mixture of n-gram, syntactic language models (SLM) and topic models (PLSA), but RNNME performs best. The out-of-domain test set shows that due to its compactness, RNNME is better suited for LM adaptation.

Tan et al. [108] showed that by crossing sentence boundaries, perplexities can be drastically reduced. Although they did not publish any results on the check set, their mixture of n-gram, syntactic language models and topic models achieved a perplexity of 176 on the test set, a 23% relative reduction compared to KN5. A similar observation was made for the SNM models by adding a feature extractor (r, s, a) analogous to regular skip-grams, but with s now denoting the number of skipped sentence boundaries `</S>` instead of words. Adding skip-`</S>` features with $r + a = 4$, $1 \leq r \leq 2$ and $1 \leq s \leq 10$, yielded an even larger reduction of 26% than the one reported by [108]. On the check set we observed a 25% reduction.

The RNNME results are achieved with a setup similar to the one in [14]. The main differences are related to the ME features (3-grams only instead of 10-grams and bag-of-words features) and the number of iterations over the training data (20 epochs instead of 10). These choices are related to the size of the training data. It can be seen from Table 4.4 that the best RNNME model outperforms the best SNM model by 13% on the check set. The out-of-domain test set shows that due to its compactness, RNNME is better suited for LM adaptation.

4.6 Follow-up Work

After our work on the One Billion Word Benchmark [86, 87, 103], Chelba et al. followed up with additional work that illustrates other attractive properties of the SNM paradigm. In [16], they showed the ease with which new features can be added to the model by using skip-grams within as well as across queries in a given search session, in conjunction with the geo-annotation available for the query stream data. Experiments on the google.com query stream showed that with session-level and geo-location context, it is possible to achieve perplexity reductions of up to 51% relative over a Kneser-Ney n-gram baseline. Both sources of context information (geo-location and previous queries in session) are about equally valuable in building a language model for the query stream.

In [15], they used a multinomial loss to tune the adjustment model on held-out data which enabled them to overcome the mismatch between training and test data. They found that fairly small amounts of held-out data (on the order of 30-70 thousand words) are sufficient for training the adjustment model and achieved perplexity reductions of up to 15%.

Finally, in [88] they presented an in-depth comparison of SNM and ME using 5-gram features and found that, although SNM achieved a slightly lower perplexity, both models achieved about the same WER after rescoring.

4.7 ASR Integration

Although we have demonstrated that SNMLMs with skip-gram features can match the current state-of-the-art RNNLMs at a serious increase in efficiency, it is clear that the long-distance skip-gram features are a bad fit for time-synchronous decoding which restricts their use in the first pass of a speech recognizer to time-asynchronous decoding. However, as SNMLMs can also be used with short-distance features, they have the potential to be converted to FSTs which means that they can be immediately plugged into existing recognition systems. In fact, in [86] we have already demonstrated how an SNMLM using n-gram features can be converted to the standard ARPA or Doug Paul back-off format. In that paper, we have also presented a simple pruning mechanism that gives fine-grained control over the model size, which makes it scalable to very large datasets.

4.8 Conclusion

We have presented SNM, a novel probability estimation technique for language modeling that can efficiently incorporate arbitrary features. A first set of empirical evaluations on two data sets shows that SNM n-gram LMs perform almost as well as the well-established KN models. When we add skip-gram features, the models are able to match the state-of-the-art RNNLMs on the One Billion Word Benchmark [14]. Combining the two modeling techniques yields the best known result on the benchmark which shows that the two models are complementary.

On a smaller subset of the LDC English Gigaword corpus, SNMLMs are able to exploit cross-sentence dependencies and outperform a mixture of n-gram models, syntactic language models and topic models. Although RNNLMs still outperform SNM by 13% on this corpus, a complexity analysis and measured runtimes show that the RNN comes at the cost of an increased training and evaluation time.

We conclude that the computational advantages of SNMLMs over both Maximum Entropy and RNN estimation promise an approach that has large flexibility in combining arbitrary features effectively and yet scales gracefully to large amounts of data.

Chapter 5

Layered Decoding for Large Vocabularies

As we mentioned in Chapter 1, the current mainstream approach to automatic speech recognition is to combine all knowledge sources, acoustic models (AMs), language models (LMs) and lexicon, into one huge search space. This all-in-one approach has considerable advantages. First, it has been well developed and proven to be a reliable method for all kinds of tasks. Second, by immediately integrating the lexicon and LM, it is able to prune the large amount of confusion in the acoustic signal based on all knowledge sources. However, the monolithic search strategy has some disadvantages as well. Integrating all knowledge sources at once makes for a complex task which means that the knowledge sources all have to be kept simple. Consequently, almost all recognizers employ non-optimal linguistic components such as static lexica (lexicalization of morphological processes) and n-gram LMs. Furthermore, since the AM operates from left to right, it enforces this mode of operation on the other models as well. This is often inefficient for decisions which (partially) depend on a right context, for example the LM. Third, only the knowledge sources that fit the Hidden Markov Model (HMM) paradigm can be readily included. Duration, prosody and cross-frame properties in general are much more difficult to exploit. Finally, when targeting Large Vocabulary Continuous Speech Recognition (LVCSR), lexica grow and LM perplexities increase and hence the impact of integrating the lexicon and LM at an early stage diminishes.

These factors were the motivation to develop a new architecture called FLaVoR (Flexible Large Vocabulary Recognition) [30] in which the decoding is split into two layers. A first layer takes care of the acoustic recognition to output a dense

phone network. The output of this layer serves as input to a second layer in which the lexicon and LM are used to do word decoding. Decoupling the two layers makes it possible to incorporate cross-frame information after phone recognition and more importantly, to integrate more complex models, such as the ones presented in Chapters 3 and 4 in the word recognition stage. This approach has been proven to match the standard all-in-one approach for the English Wall Street Journal test suite [30,32,36]. This task however was limited to read, noise-free speech and was performed using relatively small lexica (<20k words) and LMs (bigrams).

In this chapter, we show that a basic FLaVoR setup can compete with an all-in-one approach for systems with large lexica (400k words) and LMs (5-grams), and that it can handle speech that is spontaneous and noisy. The setup can then be used to exploit FLaVoR's flexibility to further improve its accuracy.

FLaVoR was especially designed to contrast current architectures by introducing more advanced linguistic knowledge at the subword (syllables, morphemes) level which makes most sense for synthetic languages i.e. languages that exhibit a high morpheme-per-word ratio. Although this is certainly the case for strongly agglutinative languages like Turkish and Finnish, it is also true for languages with a productive morphology such as German, as opposed to English where the words are more or less atomic entities and hence operating at the word level is practical. Therefore, we chose to test the system on a language that would benefit more from the FLaVoR design than English. Dutch is a morphologically productive language which uses inflection, derivation and compounding to produce new words. It also has a high rate of foreign words and a large variety of different accents. These properties make it well suited for the FLaVoR architecture.

The chapter is organized as follows. Section 5.1 describes the FLaVoR architecture in more detail. In Section 5.2, we introduce the task, explain the corresponding setup of the FLaVoR system and talk about the experiments we did for both the phone and word decoding layers. We present a thorough discussion of our results in Section 5.3 and end with a conclusion in Section 5.4.

5.1 The FLaVoR Architecture

In this section, we briefly recapitulate the FLaVoR architecture, shown in Figure 5.1. For more details, the rationale behind FLaVoR and differences with existing multi-pass strategies, we refer to [30].

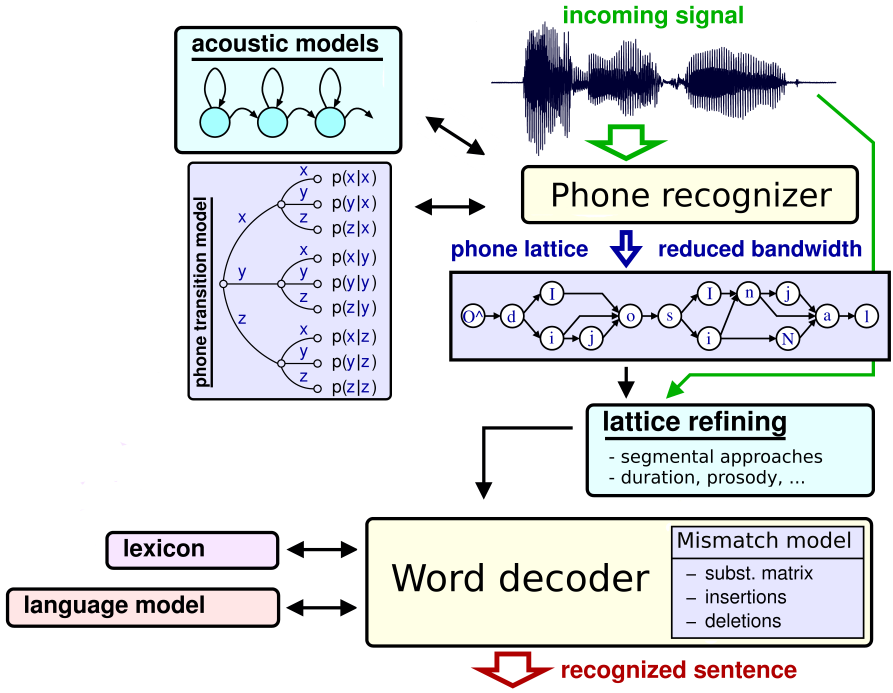

Figure 5.1: The FLaVoR architecture.

5.1.1 Layer 1: Phone Decoding

In the first layer of the FLaVoR system, a phone decoder determines the network of most probable phones given the acoustic features of the incoming signal. The knowledge sources employed are an AM and a phone transition model, i.e. a LM for phones. The resulting phone network can be enriched with meta-data such as prosody or speaker identities in order to provide rich information to the second layer. The meta-data is not restricted to the HMM paradigm which opens up opportunities to incorporate cross-frame information.

5.1.2 Layer 2: Word Decoding

The goal of the second layer of our FLaVoR system is to adopt the phone lattice as input and map phone sequences onto words without enforcing the left-to-right operation that is typical for acoustic decoding. As a prototype,

phone-to-word mapping is achieved by transforming the lexicon and LM into a Finite State Transducer (FST) and applying this to the phone lattice. It has been shown that such transducers are a very compact and efficient solution for decoding [75].

The decoupling of acoustic and word decoding not only relaxes the constraints on the mode of operation of the linguistic models, but it also reduces the number of parallel options each input stands for. While a monolithic search engine has to match all incoming feature vectors with all possible combinations of phones and end positions at that point in the search, the phone network will only contain the set of best matching phones with their optimal start and end times.

To handle the mismatch between the observed acoustics and the canonical transcription, the decoder is equipped with a mismatch model i.e. a phone confusion matrix with costs for each phone substitution, insertion and deletion. This matrix is created by retraining an initial confusion matrix based on phonetic properties on a huge corpus using Expectation Maximization (EM). The mismatch model could be incorporated in the search by means of an FST, but as was explained in [32], a dedicated implementation was chosen for efficiency reasons.

5.2 Experiments

5.2.1 Task and Reference Results

For the purpose of testing Dutch spontaneous speech the N-Best evaluation benchmark [57] was chosen. N-Best contains Northern and Southern Dutch, broadband and telephone speech, totaling to 260 hours of training data. For each of the four subtasks, the amount of speech consists of 1 to 2 hours of development data and 2 to 3 hours of evaluation data. The evaluation data is known to be different from both the training and development data in a sense that it contains considerably less telephone speech and more accented and spontaneous speech. For most systems this results in a big difference in WERs as audio normalization and adaptation were not the focus of development [31, 34,57]. As a reference we used the results of our all-in-one system [31] developed for Southern Dutch. Because the AMs for telephone speech are very different from those for broadband speech we decided to limit the task to the broadband speech for the time being. This reduced the amount of speech to 40h of training data, 55 minutes of development data and 1h55 of evaluation data.

5.2.2 Setup

In this section, we briefly discuss the models we employed in our system. For more information we refer to [31].

Acoustic Modeling

For all of the experiments, we used our in-house speech recognition toolkit SPRAAK [28]. For the AMs, 49 three-state acoustic units (46 phones, silence, garbage and speaker noise) and one single-state phone (short schwa) are modeled using our default tied Gaussian approach, i.e. the density function for each of the 4k cross-word context-dependent tied states is modeled as a mixture of an arbitrary subset of Gaussians drawn from a global pool of 50k Gaussians. The mixtures use on average 180 Gaussians to model a 36 dimensional observation vector of MIDA features [28]. These were obtained by means of a mutual information based discriminant linear transform (MIDA) on vocal tract length normalized (VTLN) and mean-normalized MEL-scale spectral features and their first and second order time derivatives (the lowest and highest MEL filter bank outputs are removed). The models were trained for an all-in-one system, i.e. the context dependent phone models are trained to cope with most of the pronunciation variation.

Language Modeling and Lexicon

The LM training material consisted of 4 main text components: 12 Southern Dutch newspapers, 10 Northern Dutch newspapers and transcriptions of broadcast news and conversational telephone speech, which together contain ca. 1 billion words. To reduce the OOV rate, caused mostly by compounds, the texts were preprocessed by a decompounding module after which 5-gram word LMs with modified Kneser-Ney smoothing were trained using a vocabulary of 400k words. The text components were interpolated linearly and perplexity minimization was done to find the optimal interpolation weights. Lexicon creation was handled by an updated version of the system described in [29]. Dutch has a decent amount of (regional) pronunciation variation which was addressed by using phonological rules to generate the likely pronunciation variants. This resulted in a median of 3.8 pronunciations per word or 1.13 variants per phone in the canonical word transcriptions.

Post-processing

Since Dutch compounds are always written as 1 word, the word recognition results were post-processed for compounding. Two subsequent words were replaced by their compound if the following criteria are met: 1) the words are longer than 3 letters, 2) the words are not very rare, 3) the unigram count of the compound is higher than the bigram count of the individual words. This approach essentially extends the 400k lexicon to a 6M lexicon.

5.2.3 Results

This section outlines the different experiments we ran for both the phone and word decoding layer. All parameters were optimized on the development data only. All timings were obtained using an Intel Core i5-2400 3.10 GHz processor with 1 core only.

Phone Decoding

To optimize a phone decoder, a reference phonetic transcription of the data is needed to test the accuracy of the system. Most databases, including N-Best, however only have an orthographic transcription, so it's necessary to convert this orthographic transcription into a phonetic one. As was shown in [29] a reliable way of doing this automatically is to create a pronunciation network by looking up the pronunciation for each word in the lexicon. Some additional language-specific pronunciation rules were applied to account for cross-word phenomena and typical word-internal assimilation processes. The Viterbi algorithm was used to find the best path through the resulting network.

Applying the same technique on the 40h of training data, a phone transition model was estimated. We created a 4-gram and tested 3 different smoothing methods: Witten-Bell (WB), Good-Turing (GT) and (modified) Kneser-Ney (KN).

Our FLaVoR system has 4 parameters to be optimized for the phone recognition layer. To combine the scores of the AM and the phone transition model we employed our standard way of handling this problem [28], by having a LM scaling factor and a word startup cost. Beam search pruning was applied to control the number of hypotheses in the network [106]: a threshold indicates how much the score of a hypothesis can drop below the score of the most likely hypothesis; if most hypotheses have a similar score, a beam width parameter

is then set to indicate how many hypotheses can be retained, keeping only the best ones.

With optimal parameters the GT smoothing achieved a phone error rate (PER) of 14.70% and was used in the following experiments, although the differences with the WB and KN smoothing were dismissible. Since it is uncertain that the best phone sequence hypothesis yields an optimal word sequence, we investigated our lattice with respect to the amount and quality of phone hypotheses it contains. Additional statistics were calculated and can be found in Table 5.1. The density of the lattice is measured as the average number of different phones (ignoring the context) in parallel per frame in the phone lattice. By relaxing the pruning parameters, lattices of different densities were made (Small, Medium, Large, eXtraLarge) to find which phone lattice density yields the best result at a reasonable speed in the second layer. To get an early indication of the quality of all hypotheses in each lattice we also calculated their lattice error rate. The lattice error rate is the PER of the path that aligns best with the reference transcription. Finally the processing time for each of the lattices was included as the real-time factor (xRT).

	S	M	L	XL
density	4.37	5.41	6.52	7.64
lattice error rate	1.53	1.24	1.09	1.00
processing time (xRT)	0.25	0.31	0.37	0.40

Table 5.1: Phone lattice statistics for the development data.

Word Decoding

We created an FST consisting only of lexicon and LM, applied it to the different lattices and optimized the different parameters of our word decoder for each of them. The parameters are the same as in the first layer of our system: score combination and beam search pruning parameters. When applying the transducer to the lattice, it becomes clear that not every sentence reaches a valid end state: if no valid word sequence can be found in the phone sequence network, there is no way for the FST to recover. In order to overcome this issue and to further improve results, we allow the decoder to substitute, insert or delete phones. In practice we train a mismatch model based on a large corpus to find typical confusions, i.e. typical mistakes the recognizer (and often also humans) makes, by comparing the output of the recognizer with the transcription. Confusions that are very common e.g. substituting the two fricatives ‘s’ and ‘f’ will be given a low cost while very unlikely confusions e.g.

substituting the vowel ‘a’ and the consonant ‘d’ will be very costly. For every possible insertion, deletion and substitution we calculate its cost to end up with a full confusion matrix containing all the costs. A *single mismatch constraint* is set to prevent the huge growth of hypotheses: after each mismatch the next phone is required to be correct. By allowing only a single phone operation in a row the recognized word sequence cannot deviate too much from the phone sequence hypotheses in the lattice. Figure 5.2 shows an example of how the mismatch model works.

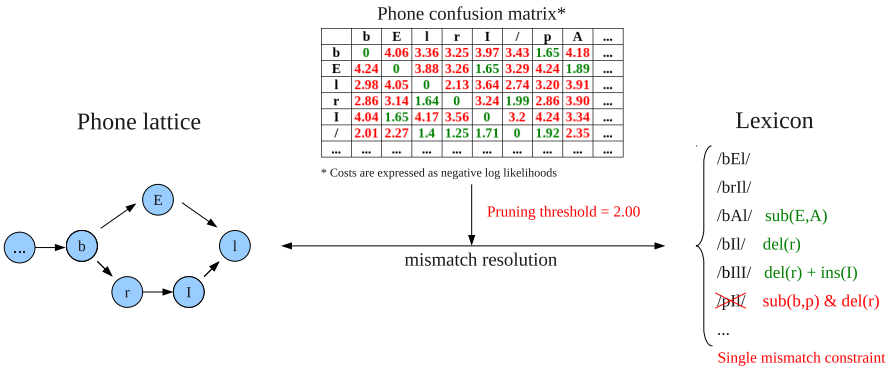

Figure 5.2: Mismatch model.

Training the mismatch model was done by creating an initial confusion matrix based on the phonetic properties of every Dutch phone. To optimize the cost of every substitution, insertion and deletion we created a phone lattice of the N-Best training corpus using the first layer of our system and then used this lattice to estimate the optimal costs, given the initial matrix and the reference transcription.

5.3 Discussion

During our experiments we found that the FLaVoR approach is very robust with regards to the phone recognition layer. Changing the task, the phone smoothing method or the preprocessing hardly has any effect on the optimal parameters of the first layer. Furthermore, as can be seen in Table 5.1, the creation of the phone lattices, which is a fixed cost, is faster than in real time, even for the XL phone lattice. This means that making changes to the phone layer is very easy and fast. Moreover, the generic nature of the first layer allows it to function in any knowledge domain for a specific language. In addition, the phone information itself could be used in certain applications (e.g. language

learning [36]), for handling specific problems (e.g. recognition of proper names) or for keyword spotting.

As depicted in Table 5.2, the FLaVoR approach without the mismatch model is very fast for both development and evaluation data, with processing times not much higher than the creation of the phone lattices. The obtained WERs with this setup are still acceptable considering the fast decoding, but incorporating some form of mismatch modeling is preferable. The WERs clearly improve when increasing the density of the phone lattice, at the cost of an increase in processing time.

When mismatch models are employed, the differences in WERs and processing times between the various phone lattices are less pronounced, especially in the development data. We believe that the current mismatch model is powerful enough to cope with a mild amount of confusability in the data, even when only a limited number of original hypotheses are included in the phone lattice. Since the evaluation data contains more spontaneous and accented speech, its confusability is considerably higher and the word decoding layer can still gain from extra phone hypotheses. Better tuned mismatch models should be able to handle the increased confusability, thus eliminating the need for higher densities and improving the processing time as well as the WER.

When comparing our WERs with the all-in-one approach we see that for each lattice the WER is competitive for both development and evaluation data. Moreover, with regards to decoding speed it should be noted that our system

		dev		eval	
		xRT	WER	xRT	WER
all-in-one		0.50	6.29	0.88	19.94
		1.38	5.71	2.57	19.16
		3.56	5.45	9.49	18.80
FLaVoR without mismatch model	S	0.41	11.08	0.68	23.61
	M	0.55	9.61	0.86	22.49
	L	0.70	8.88	1.03	21.85
	XL	0.84	8.16	1.41	21.54
FLaVoR with mismatch model	S	3.42	5.81	4.27	19.19
	M	3.70	5.81	4.55	19.15
	L	3.93	5.79	5.13	19.01
	XL	4.17	5.63	5.69	18.96
FLaVoR with pruned mismatch model		1.67	5.76	2.72	19.26

Table 5.2: WERs and processing times (xRT) for all-in-one and FLaVoR.

handles the discrepancies between development and evaluation data much better. When using an all-in-one approach the processing times almost double or even triple, while ours differ only by a factor 1.2 to 1.4. WER optimization was done for every lattice without taking decoding speed into account. For better time comparison, the mismatch model was pruned by ignoring phone operations that have too high a cost compared to a threshold value. This pruned model yields the best combined results of WER and decoding speed on the XL and L lattices for development and evaluation data respectively. Again the results are competitive, but given access to all the knowledge sources the all-in-one approach currently still has an advantage. Analysis of both decoders showed however that when it comes to investigating the different explanations according to the LM, the FLaVoR approach definitely wins, i.e. it will benefit more from more powerful LMs. Moreover, there are a lot of unexplored opportunities to further improve our system.

In all our experiments, we limited ourselves to 4-gram phone transition models. It is likely that enlarging the phone context has a positive impact on both phone and word recognition results. However care must be taken to avoid overfitting since there is only a limited amount of training data.

Another standing issue is how pronunciation variations can best be handled. In the current setup, this task is divided rather arbitrarily between the AM (the phone models were trained based on canonical lexicon pronunciations), the lexicon (pronunciation rules) and the mismatch model. Optimizing the role of each component (which component models which part of the variation) has the potential to both improve the recognition and to speed up the decoding.

It is clear that the mismatch model has a large impact on the result, both in WER and processing time. Improving this model will not only lead to lower WERs, but also to less hypotheses to consider, thus making the whole system a lot faster. The ideal model we want to approach will have only a little overhead compared to the system without a mismatch model. This will provide the opportunity to incorporate even more complex knowledge sources in the second layer.

Likely candidates for improving the mismatch model consist of conditional substitutions, insertions and deletions. One possibility would be to take phone duration into account when considering the possible phone operations e.g. the substitution of a long vowel into another one should be more costly than the substitution of a short vowel and its deletion should only be allowed at a very high cost. Optimal duration boundaries should be investigated for all phones.

Phone context is a second possible upgrade of which we believe the mismatch model and hence the WERs and times will benefit. In spontaneous speech,

human pronunciation is typically very sloppy which results a.o. in vowels, even long ones, being substituted by the neutral schwa. Our mismatch model correctly estimates this behavior, consequently assigning low costs to these substitutions. Our phone transition model however indicates that in Dutch the schwa can appear after a diphthong with a relatively high probability, while other vowels, especially long ones, are very rare if not impossible to appear in this context. We believe similar phenomena exist for larger contexts, as well as right contexts.

5.4 Conclusion

We have presented an extension of the FLaVoR architecture to large vocabularies (400k words), large language models (5-grams) and more spontaneous, noisy speech in a morphologically rich language. This layered decoder was designed to decouple phone and word recognition which allows for the integration of more complex linguistic components, especially at the sub-word level. It was tested on the Dutch language which – with its large variety of accents and rich morphology – is ideally suited to benefit from this integration. We have shown on the N-Best benchmark that, although this architecture is yet to reach its full potential, it is already competitive to an all-in-one approach in which acoustic models, language models and lexicon are all applied simultaneously.

Chapter 6

Language Model Adaptation for ASR of Spoken Translations

In the previous chapters, we have focused on efficient solutions to enable more powerful language modeling, either by proposing novel and efficient language models or by proposing a speech recognition architecture that reduces the load during the application of the language model. In this chapter, we focus on language model adaptation, more specifically in the context of computer-aided translation (CAT). We propose efficient adaptation techniques that improve the recognition accuracy by exploiting the source language text, translation models and named entity models with only minimal overhead.

The chapter is organized as follows. First, we introduce ASR of spoken translations in Section 6.1 and language model adaptation in this context in Section 6.2. Then, we describe our proposed method of integrating translation and language models efficiently in Section 6.3 and discuss some extensions to improve the accuracy in Sections 6.4 and 6.5. Finally, we validate both the efficient integration and the extensions with two experiments in Sections 6.6 and 6.7. We end with a conclusion in Section 6.8.

6.1 ASR of Spoken Translations

Although CAT is traditionally performed with keyboard and mouse, a recent study [35] has shown that the use of automatic speech recognition (ASR) as an input method may constitute a significant speed-up to translators, even with a non-perfect speech transcription that needs additional correction. Furthermore, it has been established that by using a translation model and more specifically the translation probabilities of the words and/or word groups (phrases) of the source language text, the speech recognition of spoken translations can be improved [11, 58, 83, 91, 95].

There are several different scenarios that can be applied to combine models for decoding the optimal transcription of a spoken translation where each implies different assumptions about system implementation and constraints on computational complexity. All of them are based on a Bayesian extension of the ASR maximum likelihood formula, first proposed by [11]: given a source language text $F = f_1, \ldots, f_I$ and an acoustic signal $X = x_1, \ldots, x_T$, which is the spoken version of a target language text $E = e_1, \ldots, e_J$, the optimal transcription $\hat{E}$ is decoded as follows:

$$\begin{aligned} \hat{E} &= \arg\max_E P(E|X,F) \\ &= \arg\max_E P(X,F|E)P(E) \\ &= \arg\max_E P(X|E)P(F|E)P(E) \end{aligned} \tag{6.1}$$

where the conditional independence assumption of X and F given E is considered to be reasonable and allows a decomposition into three knowledge sources: the acoustic model (AM) $P(X|E)$, the translation model (TM) $P(F|E)$ and the language model (LM) $P(E)$.

One scenario in which this extension can be used is to rescore hypotheses generated by the ASR system, using the TM probabilities as part of a multi-pass approach. This can be done either by re-ranking ASR N-best lists [10, 59, 83] or by rescoring word lattices [58, 91]. One of the main issues with multi-pass approaches however, is that the recognizer does not have access to machine translation (MT) information during the first pass. The recognizer might have already pruned out several interesting hypotheses that no rescoring can recover. Another issue is that the output of the recognizer has to be stored and that the second pass can only start when the first pass has finished, which takes up valuable space and time. In human-computer interaction, response times and storage should be minimized to reduce the overhead associated with rescoring.

6.2 Language Model Adaptation

Only very recently did the multi-pass scenario make room for a new scenario which we believe will be the new paradigm in MT-ASR integration. Rodríguez et al. [95] show that, for each source language sentence, the language model of the ASR system can be updated using information derived from the text of that individual sentence. This approach is claimed to yield a decrease in computational complexity by efficiently pruning the LM as well as a significant reduction in word error rate (WER).

Instead of applying a multi-pass approach, the authors propose a direct integration of the translation model probabilities into the ASR language model. They do this by approximating $P(F|E)$ in Eq. (6.1) at the sentence level, only taking into account lexical translation probabilities. That is, each word e_j in the target language sentence E corresponds to exactly one word f_i in the source language sentence F:

$$P(F|E) \approx \prod_{j=1}^{J} \max_{i:f_i \in F} P(f_i|e_j) \tag{6.2}$$

The product of $P(F|E)$ and $P(E)$ can then be considered as the new language model $P'(E)$ with which speech is decoded.

If we consider an n-gram to be a tuple (h, e) consisting of a history h and a target word e, then adapting the n-gram probability $P(e|h)$ to $P'(e|h)$ boils down to:

$$P'(e|h) = \frac{P(e|h) \max_{i:f_i \in F} P(f_i|e)}{norm(h)} \tag{6.3}$$

The normalization factor $norm(h)$ is obtained as follows:

$$norm(h) = \frac{\sum_{e':(h,e') \in \mathcal{T}} P(e'|h) \max_{i:f_i \in F} P(f_i|e')}{\sum_{e':(h,e') \in \mathcal{T}} P(e'|h)} \tag{6.4}$$

where $\mathcal{T}$ corresponds to the training data used to train the original language model probabilities $P(e|h)$.

Once all the relevant probabilities have been updated, the back-off weights $bow(h)$ are also renormalized to obtain a true probability distribution $P'(e|h)$:

$$bow(h) = \frac{1 - \sum_{e':(h,e') \in \mathcal{T}} P'(e'|h)}{1 - \sum_{e':(h,e') \in \mathcal{T}} P'(e'|h')} \tag{6.5}$$

where h' denotes the history after back-off.

In addition to these updates, the authors also allow out-of-vocabulary (OOV) named entities to pass through and be included untranslated into the target language LM and the ASR lexicon. The named entities are assigned a unigram probability based on the length of the target sentence.

The authors claim that their suggested approach is efficient, because only the relevant n-grams (h, e), i.e. the n-grams for which the TM contains a probability $P(f|e)$, need to be updated. It is not clear however what happens to the irrelevant n-grams, i.e. the ones for which $P(f|e)$ is absent from the TM. Not updating these n-grams corresponds to multiplying the original probability $P(e|h)$ by 1 which is always larger than or equal to $\max_{i:f_i \in F} P(f_i|e)$, leading to relatively larger probabilities for irrelevant n-grams. It would be more logical to decrease the probabilities of the irrelevant n-grams e.g. by multiplying with a small value $\epsilon \ll 1$, but then each n-gram in the model would need updating, which would have a negative impact on efficiency.

Even if we assume that the authors apply a smart technique to solve this issue, their approach still requires computing the normalization factor and back-off weight for each history h. In a real-time application, this extra computation results in increased latency and could potentially harm the translator's efficiency which is exactly the opposite of what we are trying to achieve. Moreover, even when the source text is known beforehand and a new LM can be trained and stored for each sentence in advance, the approach requires a lot of storage and the speech recognizer needs to switch language models every time. In the next section, we will describe an alternative to this approach that overcomes all of these issues.

6.3 Efficient Integration of Translation and Language Models

6.3.1 Doing Away with Normalization

A first and important observation for our proposed method is that LM normalization is not mandatory in the context of ASR decoding. In general, statistical language models assign probabilities to words and word sequences according to a certain probability distribution. If this were not the case and language model scores would be allowed to have any value, then a measure such as perplexity that compares LMs based on the score given to each observed word would lose its meaning entirely. It is therefore necessary to renormalize a language model whenever some of its probabilities are updated. In back-off n-gram LMs, renormalization not only consists of computing a normalization

factor for each history, but also requires recomputing the back-off weight for that history, making it a relatively complex operation, as shown by Eqs. (6.4) and (6.5).

In ASR decoding however, a score is not attributed to a single word sequence, but rather to many competing word sequence hypotheses. Therefore, there is no strict constraint that the score that a LM attributes to each hypothesis should obey a true probability distribution, as long as a more likely hypothesis receives a higher score. Moreover, competing word sequence hypotheses E are not simply given a score according to their acoustic likelihood $P(X|E)$ and language model prior $P(E)$, but typically also include a LM scaling factor a and word insertion cost c:

$$score(E) = P(X|E)P(E)^a c^J \tag{6.6}$$

where J denotes the length of word sequence E. The LM scaling factor compensates for the acoustic model's probability underestimation due to feature correlation and the overlarge dimensionality that acoustic features typically exhibit compared to the true dimensionality of speech [1]. That is, the acoustic model probabilities are generally a lot smaller than the language model probabilities. The LM scaling factor balances these two probabilities by decreasing the language model probabilities. However, this has the unwanted side effect of increasing the dynamic range of the language model which means that it is now more biased towards the more probable short words than before. The word insertion cost is used to overcome this bias and prefer hypotheses with less, hence longer words. The inclusion of these two parameters makes the final score that is attributed to a word sequence E unnormalized, even if $P(X|E)$ and $P(E)$ are normalized. Note however that, although this means that the LM score does not have to be normalized, it is still important that the three factors in Eq. (6.6) operate on a similar scale if we want them to have a comparable impact on the final score.

6.3.2 Probability Inflation

As was explained in Section 6.2, updating LM probabilities by multiplying with TM probabilities, requires updating all n-gram probabilities $P(e|h)$, even those for which $P(f|e)$ is absent from the TM. This is due to the deflating nature of the update: if we instead were to multiply the relevant n-gram probabilities by values larger than 1, the irrelevant ones could remain untouched, thus significantly reducing the number of updates. We therefore propose an alternative update rule that essentially inflates rather than deflates

the n-gram probabilities:

$$score_{LM}(e|h) = P(e|h) \max_{i:f_i \in F} g(f_i, e) \tag{6.7}$$

where g is a weighting function that maps TM probabilities $P(f|e)$ onto values larger than 1. To have some control over the shape and maximum of this function, yet minimize the introduction of new parameters, we propose the following exponential function:

$$g(f, e) = 1 + \alpha\beta^{1-P(f|e)} \tag{6.8}$$

where $\alpha \in \mathbb{R}_0^+$ controls the maximum value of the update weight and $\beta \in]0, 1[$ determines the relative weight that is given to $P(f|e)$: a smaller value of β will give a relatively higher weight to high probabilities than to low probabilities. This essentially increases the dynamic range which can be useful if the translation model is unsure and assigns low probabilities to many words. It is important that these parameters are carefully optimized on held-out data, because if we inflate the LM probabilities too much we run the risk of overshadowing the acoustic score or the word insertion cost and ending up with acoustically implausible hypotheses or hypotheses consisting only of short words.

6.3.3 Applying Inflation Weights

Rodríguez et al. [95] create a new LM for each sentence, which means that – if the LM cannot be computed in real time – the new LM has to be stored and loaded for each utterance. For tasks that involve a lot of sentences and large LMs, this puts a serious load on the system that runs the software to assist humans in translation and may result in a translator being less rather than more productive.

To prevent such an unwanted scenario, we do not store each updated LM in its entirety, but instead store only the inflation weights. This seriously reduces the storage cost and it allows the speech recognizer to limit memory loading to a small set of inflation weights for each sentence.

6.4 Phrase-based Models

Although the implementation described in Section 6.3 improves the efficiency of MT-based LM adaptation, Eq. (6.2) assumes translation consists solely of one-to-one alignments i.e. each word f_i in the source language text can only

correspond to one word e_j in the target language text. This is a strong assumption that does not hold in reality: every language has its own way of verbalizing concepts with some using a single word and others using multiple words for the same concept. In machine translation this issue is addressed by so-called phrase-based translation models [63]. In the next sections, we will explain in more detail what phrase-based translation entails and how we can use it to extend our adaptation scheme and improve the recognition accuracy.

6.4.1 Phrase-based Translation

Phrase-based translation models are models that cover correspondences between sequences of words, called phrases, which we denote by $\bar{f}$ and $\bar{e}$ for source and target language, respectively. This approach has several advantages. It allows for translating a longer sequence to a shorter sequence or vice versa. For instance, English *grey horse* translates to Dutch *schimmel*, and the English compound *screen resolution* to *schermresolutie*. It allows for capturing local context. For instance, *large horse* can be translated word by word to Dutch (*groot paard*), but the combination of *grey* and *horse* should be translated as a whole, as it (most likely) indicates a specific type of horse for which there is a specific word in Dutch. It should be stressed here that a phrase is not a linguistic notion in a phrase-based MT model: the sequence *looks at the* (*kijkt naar de*) may be a phrase, although it is not a linguistic constituent. This example sequence also shows that phrases capture the context of words which have many possible translations, like prepositions. In the phrase *looks at the*, the word *at* should be translated into Dutch as *naar*; compare this to *laughs at the* (*lacht om de*) where *at* is translated as *om*. Figure 6.1 shows a comparison of word-based and phrase-based English-to-Dutch translation of the sentence *leaders have to suffer*.

Correspondences between phrases are constructed from word-aligned sentence pairs, and stored in a phrase table, i.e. a list of phrase pairs with associated scores. This is a three-step process, the details of which are discussed in the below paragraphs. In the first step, the two word alignments (one for each language direction) of a sentence pair are turned into a single, symmetrized one. In the second step, phrase pairs are extracted from the symmetrized word alignment of a sentence pair. In the third step, phrase pair scores are calculated based on the full set of phrase pairs extracted from all sentence pairs.

The first step in building a phrase table consists of combining the two word alignments of each sentence pair. The first alignment links source words to target words (a source word may be linked multiple times, i.e. have links with more than one target word). The second alignment links target words to

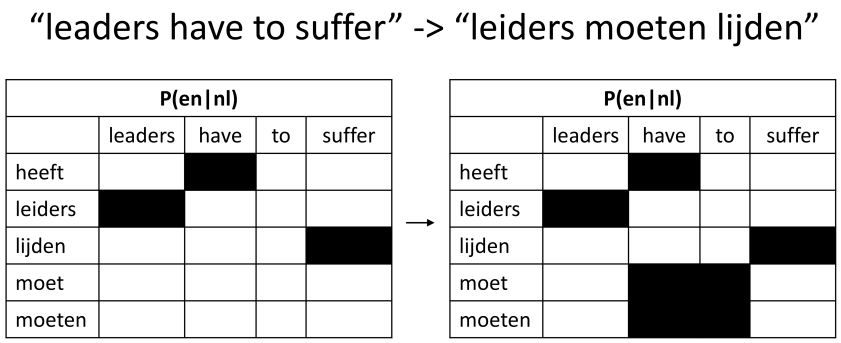

Figure 6.1: Comparison of word-based and phrase-based translation. The phrase-based alignment is able to detect that *have to* is a phrase which is translated in Dutch by the single word *moet* or *moeten* (correct in this context).

source words (multiple linking is also possible here). The two word alignments are symmetrized, which can take place in several ways. For instance, if symmetrization only keeps links which occur in both alignments, we obtain an intersective word alignment. Another type of symmetrization uses a heuristic to add additional alignment points to the intersective alignment. From the symmetrized alignments of all sentence pairs, lexical probabilities $w(f|e)$ and $w(e|f)$ are estimated by relative frequency:

$$w(f|e) = \frac{C(f,e)}{\sum_{f'} C(f',e)} \tag{6.9}$$

where $C(f,e)$ stands for the number of times f aligns with e in the training data. These lexical probabilities express the likelihood that one word is translated by another one[1]. Their use is further detailed below.

The second step in the phrase table construction procedure consists in extracting consistently aligned phrase pairs from the symmetrized alignment of a sentence pair. These are phrase pairs in which each source word is either aligned to a word in the target phrase, or not aligned at all, and in which each target word is either aligned to a word in the source phrase, or not aligned at all.

[1]Before calculating these probabilities, a NULL token is added to the target sentence, which is aligned to each source word that was not aligned to a target word. This allows for calculating the probability that a source word is not translated. The same is done in the other direction.

The third and final step in creating the phrase table consists of calculating four scores for each phrase pair $(\bar{f}, \bar{e})$. The first score is the phrase translation probability $\phi(\bar{f}|\bar{e})$, which is the phrase equivalent of the lexical probability $w(f, e)$ and is also estimated by relative frequency. The second score is the lexical weight, which validates the quality of a phrase pair by checking how well its words translate to each other. This serves to compensate for an overestimation of the reliability of rare phrase pairs. Given a phrase pair $(\bar{f}, \bar{e})$ and a word alignment a between the target language word positions $j = 1, \ldots, m$ and source language word positions $i = 1, \ldots, n$, the lexical weight P_w is computed as the product of the average lexical probabilities $w(f|e)$ of the phrase:

$$P_w(\bar{f}|\bar{e}, a) = \prod_{i=1}^{n} \frac{1}{|\{j : (i, j) \in a\}|} \sum_{\forall (i,j) \in a} w(f_i|e_j) \tag{6.10}$$

The third and the fourth score are inverses of the first and second one, i.e. they apply to the translation in the other direction.

The four phrase pair scores described above are combined by the MT system using a log-linear model, which also involves other scores, such as a language model score, a reordering score, ... We will not discuss these other scores further, as they are not relevant to the combination of phrases and ASR described here. The weights for the scores are determined during a tuning process where the MT output for a set of held-out source sentences is compared to their reference translations. Finally, when the MT system translates a source sentence F, it selects the target sentence E with the highest score according to the log-linear model.

6.4.2 Phrase-based Adaptation

By using phrase-based translation models, we can extend Eq. (6.2) as follows:

$$P(F|E) \approx \max_{a, \bar{f} \in F, \bar{e} \in E} \prod_{j=1}^{l} P(\bar{f}|\bar{e}_j) \tag{6.11}$$

where l denotes the number of phrases in E, given the alignment a that maximizes $P(F|E)$.

$P(\bar{f}|\bar{e})$ can then be estimated by combining the four phrase pair scores described in Section 6.4.1. Because the update weights in Eq. (6.8) are based on probabilities and the normalization of a log-linear model is expensive, we

opted to combine the scores via linear interpolation:

$$P(\bar{f}|\bar{e}) = \lambda_1\phi(\bar{f}|\bar{e}) + \lambda_2 P_w(\bar{f}|\bar{e}) + \lambda_3\phi(\bar{e}|\bar{f}) + \lambda_4 P_w(\bar{e}|\bar{f}) \quad (6.12)$$

where the λ_i's are estimated empirically.

The weighting scheme allows us to reduce the number of updates, thus maintaining efficiency, provided that the amount of phrases to be evaluated is not too large. We will come back to this in Section 6.7.

6.5 Named Entity Models

The way named entities are written or pronounced is often similar and sometimes even identical across multiple languages, especially if the languages belong to the same language family. This means that, even if the translation model does not contain a translation for a given named entity, there is a relatively high probability that the named entity can just be copied from the source language to the target language. Choosing a value for the named entity translation probability $P(NE_f|NE_e) \approx 1$ then allows updates to n-grams containing this named entity, provided that it exists in the ASR language model and lexicon.

If however the named entity does not occur in the ASR language model and lexicon, there are no n-gram probabilities to update and we have to resort to other estimation techniques. One simple estimation technique that we have found to work well is to map new named entities in the language model to a special token `<UNK>` for OOV words. The n-gram probabilities for this special token are estimated on words that occur in the training data, but are excluded from the vocabulary. The named entities can then use the OOV probabilities, optionally weighted by a factor h_{NE}. If the pronunciation of the named entity is not too different in the source language and the target language, and the target-language grapheme-to-phoneme (G2P) converter is able to approximate the pronunciation in the source language, this very rough LM estimate often steers the recognizer into the right direction. Note that it may also be possible to use a source-language G2P, provided that both G2P's use the same phoneme set and that the pronunciations of named entities are similar in both languages.

6.6 Experiment 1: Efficient Adaptation

In this first experiment, we focus on efficient LM adaptation for spoken translations. We investigate the effect of the adaptation on disk storage and execution speed and verify whether the improved efficiency does not come at the cost of decreased recognition accuracy.

6.6.1 Task and Setup

The ASR experiments were performed on a test corpus of audio fragments from the Flemish part of the Corpus Spoken Dutch (CGN) [81], component o. The chosen fragments correspond to 167 Dutch utterances that are read translations from English books.

Using a vocabulary of 100k words, an initial 3-gram LM with modified Kneser-Ney smoothing [17, 61] was trained by running the SRILM toolkit [107] on a collection of normalized newspaper texts from the Flemish digital press database Mediargus which contains 1104M word instances (tokens) and 5M unique words (types). The vocabulary was converted into a phonemic lexicon using an updated version of the Dutch G2P described in [29] and integrated into the recognizer described in [31], which was built with our in-house speech recognition system SPRAAK [28].

The TM was created by applying the GIZA++ toolkit [80] to a set of 1M English-Dutch parallel sentence pairs extracted from the Europarl corpus [62], which contains the written version of speeches of members of the European Parliament. GIZA++ adopts an EM approach to learning lexical probabilities $P(f|e)$ and $P(e|f)$ from a parallel corpus. The approach initializes the lexical probabilities using a uniform translation distribution for words. Based on this initialization, the probabilities of possible word alignments of sentence pairs are calculated. The new probabilities then allow to recalculate the lexical probabilities across the set of sentence pairs.[2] This process continues until convergence.

To build the updated LMs, we excluded all updates for source language words shorter than 4 letters, as we found that these tend to be unreliable. An exception was made for words that start with a capital letter which are assumed to be named entities if they have at least 2 letters. Named entities that are not

[2]For the sake of clarity, we would like to point out that these probabilities are different from the ones that are calculated from word alignment in the first step of phrase table construction.

in the TM, but are in the ASR lexicon, were given an optimized translation score of 1.2.

In this first series of experiments, we only tested the influence of the word-based translation model. No new words were added to the lexicon. The inflation weights were generated with optimal values of $\alpha = 9$ and $\beta = 0.005$. Further optimization of the LM scaling factor a and word insertion cost c proved to be unnecessary.

As far as we know, there is no available implementation for the normalized models presented in Section 6.2, so we had to revert to own implementation for the LM updates according to Eqs. (6.3) and (6.4). We implemented a model that uses a small, optimized value of $\epsilon = 0.001$ for the irrelevant n-gram probabilities, as discussed in Section 6.2. We also optimized the LM scaling factor a and word insertion cost c, which in contrast to the unnormalized model lead to further improvement. Renormalizing the back-off weights was done using the SRILM toolkit.

6.6.2 Disk Storage

Using SPRAAK, the initial 3-gram LM was converted into a compressed binary format reducing its size from 2.9GB to ca. 500MB. Since we do not add any words or n-grams to the model during the update, the normalized models are the same size for each sentence. Our test set contains 167 sentences, yielding a total disk storage of 484GB for the uncompressed models or 84GB after compression. For LMs where $n = 4$ or $n = 5$, this goes up to compressed models of 1GB and 1.5GB per sentence, respectively.

By contrast, the size of the file containing the update weights for a single sentence never exceeds 250KB, yielding a total disk storage of 42MB. Moreover, as the update weights do not depend on the size of the original n-gram LM, the disk storage does not increase for higher order models. That means that for 5-gram LMs, the disk storage is reduced to 2.8% compared to a normalized model (42MB vs. 1.5GB).

6.6.3 Execution Speed

Both the normalized and unnormalized models must undergo several steps before they can be applied during ASR. For the normalized model, we need to compute the unnormalized probabilities and renormalize both the probabilities and the back-off weights. For the unnormalized model, we only need to compute the update weights. Once the models are created, the ASR system needs to

	Normalized LM		Unnormalized LM	
	sentence	total	sentence	total
load AM and lexicon	0.4s	0.4s	0.4s	0.4s
load LM	5s	5s	5s	5s
total offline	5.4s	5.4s	5.4s	5.4s
update & normalize LM	16s	44m32s	-	-
compute update weights	-	-	0.2s	34s
load updated LM	5s	13m55s	-	-
load update weights	-	-	0.01s	1.7s
total adaptation	21s	58m27s	**0.21s**	**35.7s**
total	26.4s	58m32.4s	**5.61s**	**41.1s**

Table 6.1: Execution times for the different LM steps involved in recognizing 167 spoken, translated sentences, using a normalized vs. unnormalized 3-gram LM. Times were measured on a dedicated machine with an Intel Core i5-2400 3.10 GHz processor, using a single core.

switch LMs for every sentence. In Table 6.1, we give an overview of the time it takes to perform each of these tasks for 3-gram LMs, where the reported times were acquired by averaging the times of hundreds of iterations for each step on a dedicated machine with an Intel Core i5-2400 3.10 GHz processor, using only a single core. To be fair, we do not report individual times for both normalization steps. An efficient implementation would combine the two steps, requiring only a single pass through the data. We therefore report a total normalization time that is limited to the time needed by the SRILM toolkit to recompute the back-off weights. We illustrate both the execution time per sentence and the total time to reflect the fact that some of the tasks need to be performed multiple times whereas others only happen once. Note however that for a real-time application, where a translator typically operates sentence by sentence, not all of the resources have to be available at the same time. As such, the most correct way to compare the two models is by looking at the execution time per sentence.

When considering the time needed to compute the updated LM on the fly, Table 6.1 shows that our unnormalized model is 15.8s faster than the normalized model. This is due to the fact that normalizing a LM is a lot more complex than computing update weights. As the on-the-fly computing time of the normalized model does not allow its use in a real-time application, we also investigated how much time it takes to load a precomputed model from disk for each sentence. Besides the obvious storage disadvantage that comes with this approach, it also still requires 5s to switch LM per sentence which is bound to have a negative effect on the translator's efficiency.

The unnormalized model on the other hand needs only 0.2s to compute the update weights which already allows its use in a real-time application. If lower latency is required, the same precomputing strategy can be applied which results in a loading time of only 10ms per sentence. As was shown in Section 6.6.2, the storage cost that comes with this approach is very low with only 250KB per sentence, independent of the order of the LM.

We are aware that execution speeds depend heavily on implementation and although we have attempted to use efficient implementations for each task, the reported times can differ significantly with different hardware and/or software. Nevertheless we are confident that our unnormalized model is vastly more efficient, because several tasks have been made superfluous.

6.6.4 Recognition Accuracy

Although the focus of this experiment is on efficiency and not on accuracy, it is crucial to verify that our proposed model does not come at the cost of a significantly increased WER. Table 6.2 shows that the weighted updates of our model reduce the WER by more than 5% absolute and 20% relative, compared to the initial LM that does not have any MT information at its disposal. The proposed model also outperforms our implementation of the normalized model, but we would like to stress that this implementation does not correspond exactly to the one in [95]. The modest 5% WER reduction by our implementation compared to the 30% WER reduction that was reported in [95] may be explained by the different language pairs (English-Dutch vs. French-English) and test sets. We can therefore not claim that our proposed technique is more accurate than [95], but instead conclude that it is competitive and more efficient.

6.7 Experiment 2: Accurate Adaptation

In the first experiment, we investigated the efficiency of our LM adaptation technique and found that our implementation exhibited virtually no adaptation overhead, enabling its use in a real-time setting. In this experiment, we investigate whether we can improve the recognition accuracy by employing phrase-based models and named entity models without sacrificing the achieved efficiency.

	WER	Reduction
Initial LM (no TM updates)	25.96	-
Normalized adaptation: word-based	24.61	5.2%
Unnormalized adaptation: word-based	20.68	20.3%
Unnormalized adaptation: phrase-based	20.43	21.3%
+ named entities (IV)	20.19	22.2%
+ named entities (IV+OOV)	**19.39**	25.3%

Table 6.2: WERs (in %) and relative reductions using the investigated types of adaptation, compared to an initial 3-gram model that does not use TM probabilities. A distinction is made between in-vocabulary (IV) and out-of-vocabulary (OOV) named entities. The models are evaluated on 167 utterances from the Dutch CGN corpus (component o), corresponding to translations from English.

6.7.1 Task and Setup

To compare the influence of the phrase-based models and named entity models, the task of the second experiment is the same as the first experiment. Adaptations were made to the same baseline language models with the same vocabulary. The recognition system was also unchanged.

The phrase table was created using the process described in Section 6.4.1 (the first step added additional alignment points after calculating the intersection) and filtered to contain only one-to-one and many-to-one alignments. Though many-to-many alignments could in principle yield further improvements, this comes at the cost of efficiency as the many-to-many alignments take up more than 90% of the phrase table. The final phrase table contains 2,939,355 phrase pairs which is ca. 1.6 times the size of the word-based translation model. As the computation of the update weights took ca. 0.2s per sentence for the word-based model, the overhead introduced by using a phrase-based model is negligible.

Named entities that are not in the ASR lexicon were added to the lexicon using the Dutch G2P; their LM probabilities correspond to the OOV probabilities, weighted by $h_{NE} = 1.65$ which was optimized empirically. Best results were achieved with phrase translation probabilities $\phi(\bar{f}|\bar{e})$ only i.e. $\lambda_1 = 1$, $\lambda_2 = 0$, $\lambda_3 = 0$ and $\lambda_4 = 0$. The inflation weights based on these probabilities were generated with optimized values of $\alpha = 16$ and $\beta = 0.0005$.

6.7.2 Results

Table 6.2 shows that the use of phrase-based rather than word-based translation models reduces the WER by 1% relative. We suspect that this moderate reduction is in part due to filtering out many-to-many alignments in favor of adaptation efficiency. On the other hand, it is probably also related to the nature of the test data: written stories often use complex structure and vocabulary which causes the translations to deviate substantially.

Simulating translation probabilities for English named entities that do not occur in the translation model, but do occur in the Dutch ASR lexicon and LM, also has a small, but significant effect on the recognizer. This confirms our intuition that the translation can just be copied from the source language to the target language, which is helped by the fact that in our case the source and target language are both Germanic languages.

Finally, we observe that even English named entities that are unknown to both the MT and ASR system, can be modeled well by using a Dutch G2P for the pronunciation and weighted OOV probabilities for the language model. Although it is to be expected that adding relevant words to the recognizer yields improvement, it is interesting that this constitutes the largest improvement of the investigated techniques, giving a total relative WER reduction of 6.2% compared to the word-based model and 25.3% over the unadapted 3-gram baseline.

6.8 Conclusion

We have presented an efficient language model adaptation technique for automatic speech recognition of spoken translations. The technique consists of n-gram probability inflation using exponential weights based on translation model probabilities which reduces the number of updates. It does not enforce probability renormalization and reduces data storage and memory load by storing only the update weights.

We have experimentally validated that when used with word-based translation models, this technique is capable of reducing word error rates by 5% absolute and 20% relative on spoken Dutch translations from English, while having little to no negative effect on recognition time. Compared to a normalized model, the model takes up only 2.8% of disk space for 5-grams and dramatically reduces the execution time.

We also investigated the effect of a phrase-based translation model and named entity probability estimation and found that together they achieve a relative WER reduction of 6.2% over a word-based LM adaptation technique and 25.3% over an unadapted 3-gram baseline. Moreover, the extensions come with the same efficiency benefits as the word-based model which allow their use in a real-time CAT environment. To our knowledge this is the first MT-based language model adaptation technique using a phrase-based translation model.

Chapter 7

Conclusion

This chapter concludes the thesis with a concise review of the original contributions of this work and some directions for future research.

7.1 Original Contributions

Compound-head clusters for Dutch n-gram language models

We have presented a novel clustering algorithm that clusters Dutch compound words together with their head. We have shown that compounds are well represented by their head, both semantically and syntactically, and that because of this, the clusters are a valuable source of information to improve n-gram probability estimates. Incorporating compound-head clusters in a class-based n-gram language model not only allows the addition of new words to the speech recognizer's vocabulary without introducing any computational overhead, it can also be used in a more general manner by clustering already observed words, thereby exploiting the aggregated statistics to acquire more reliable estimates. Both scenarios were validated experimentally on the Corpus Spoken Dutch.

Long-distance language model based on the continuous skip-gram

We have presented a novel semantic language model that is conditioned on a weighted average of the embeddings of the words in the history, as learned by the continuous skip-gram model. We have performed an extensive analysis of its parameters and compared it to two well-known long-distance models: cache

models and models based on Latent Semantic Analysis (LSA). Our analysis on excerpts from the Flemish digital press database Mediargus has shown that the proposed model, when interpolated with an n-gram language model, not only outperforms the two other models, but is also significantly more efficient than the LSA model.

Sparse non-negative matrix language models

We have presented Sparse Non-negative Matrix (SNM) estimation, a novel probability estimation technique for language modeling that can efficiently incorporate arbitrary features. We have shown that SNM language models trained with n-gram features are a close match for the well-established Kneser-Ney models. The addition of skip-gram features yields a model that is in the same league as the state-of-the-art recurrent neural network language models, as well as complementary: combining the two modeling techniques yielded the best known result on the One Billion Word Benchmark. On the LDC English Gigaword subcorpus, further improvements were observed using features that cross sentence boundaries. The computational advantages of SNM estimation over both maximum entropy and neural network estimation are probably its main strength, promising an approach that has large flexibility in combining arbitrary features and yet scales gracefully to large amounts of data.

Layered decoding for large vocabulary continuous speech

We have presented an extension of the FLaVoR architecture to large vocabularies, large language models and spontaneous speech in a morphologically rich language. This layered decoder was designed to decouple phone and word recognition which allows for the integration of more complex linguistic components, especially at the subword level. It was tested on the Dutch language which – with its large variety of accents and rich morphology – is ideally suited to benefit from this integration. We have shown on the N-Best benchmark that, although this architecture is yet to reach its full potential, it is already competitive to an all-in-one approach in which acoustic models, language models and lexicon are all applied simultaneously.

Efficient language model adaptation for ASR of spoken translations

We have presented a novel efficient language model adaptation technique that can be applied to the automatic speech recognition of spoken translations. The technique consists of n-gram probability inflation using exponential weights based on translation model probabilities which reduces the number of updates. It does not enforce probability renormalization and reduces data storage and memory load by storing only the update weights. We have experimentally validated that when used with word-based translation models, this technique achieves a significant word error rate reduction on spoken Dutch translations

from English, while having little to no negative effect on recognition time. We have also shown that a phrase-based translation model, combined with named entity probability estimation achieves an even larger reduction and comes with the same efficiency benefits as the word-based model which allow their use in a real-time computer-aided translation environment.

7.2 Directions for Future Research

Compound-head clustering

Although we believe that the idea of compound-head clustering should extend to other languages that, like Dutch, have a lexical morphology with concatenative and right-headed compounding, it remains to be proven that this is actually the case. It would also be interesting to investigate whether our technique can be extended to handle languages with a different lexical morphology. For example, Romance languages are typically left-headed, applying the prepositional scheme mentioned in Section 2.1.1. In these languages, head mapping could potentially improve the prediction of the words following the compound instead of the compound itself. Whether the targeted language is similar to Dutch or not, it is clear that the clustering algorithm in its current form is language dependent which limits its potential. We are therefore of the opinion that the main priority in compound-head clustering is to develop automated, data-driven compound-head clustering algorithms.

Class-based n-grams are a natural choice to take advantage of the useful source of information that are compound-head clusters. However, they are not the only language model into which clusters can be integrated. For example, it would be interesting to investigate whether they can also be successfully applied to more advanced language modeling techniques such as neural network language models.

Semantic language models

Although we did a thorough study of cache models, LSA and CSM and their properties, a lot of aspects still remain unexplored. As we mentioned, document boundaries were not marked in our training data, but instead a document was assumed to consist of 30 sentences. It would be interesting to see what the effect is of actual document boundaries or various, analogous assumptions. Moreover, we only employed decay on cache models, but this could certainly also be tested on LSA and CSM. We are also curious whether and if so, how much the CSM-based model can be made even more efficient by precomputing similarity probabilities. Finally, and perhaps most importantly, we believe more

compositionality is the next step in models using word embedding vectors. The assumption that a sentence is equal to the sum of its individual words is too simplistic as it completely ignores word order and inter-word relations.

Sparse non-negative matrix language models

Sparse non-negative matrix language models are an exciting new paradigm in language modeling with a lot of unexplored potential. In this work, we have only investigated a limited set of features: n-grams and skip-grams. It would be interesting to explore richer features similar to [43], as well as richer meta-features in the adjustment model. Another idea is to focus on improved generalization. Rather than using one-hot target vectors which emphasizes fit, it might be possible to use low-dimensional word embeddings instead. This would most likely yield a smaller model with improved generalization.

Layered decoding

In its current form, the layered decoding is still using the rather outdated GMM/HMM models. One improvement would be to upgrade the acoustic models to deep neural networks. Another improvement was already mentioned in Chapter 5 and consists of conditioning the mismatch model on e.g. duration, context or confidence score. Finally, given that the decoder is no longer constrained to the left-to-right operation of the acoustic model, it might be possible to do island parsing, decoding from one island of high confidence to the next. In combination with a more advanced language model, possibly on the subword level, we believe this has the potential to become the new state of the art.

Efficient language model adaptation for ASR of spoken translations

Although we have shown that adaptation using phrase-based translation models improved upon word-based models, the word error reductions were less than expected. We assume this is mostly because we filtered out many-to-many phrase translation pairs. It would be interesting to investigate whether it is possible to calculate phrase translation probabilities on the fly. This may allow us to retrieve and select many-to-many phrase pairs in a reasonable amount of time.

Appendix A

Meta-feature Pseudocode

The idea behind meta-features is that features often have certain properties, meta-features, that have a potential relationship to the prediction problem. Estimating weights on the meta-feature level rather than the input feature level therefore enables features that share these properties to also share weights which improves generalization. Meta-features are extracted from feature-target pairs (f, t) and can either be elementary meta-features or conjunctions of elementary meta-features. For a 4-gram feature `the quick brown` and target `fox`, the elementary meta-features are:

- feature identity = `[the quick brown]`
- feature type = 4-gram
- feature count = $C_{\texttt{[the quick brown]}*}$
- target identity = `fox`
- feature-target count = $C_{\texttt{[the quick brown] fox}}$

An example of a conjunction is the feature-target identity (`[the quick brown]`, `fox`), which is acquired by joining the feature identity and the target identity.

The meta-features weights $\boldsymbol{\theta}$ are stored in a flat hash table of predefined size. Strings are fingerprinted (converted into a byte array, then hashed), counts are hashed, and the resulting integer is mapped to an index by taking its value modulo the predefined $size(\boldsymbol{\theta})$. Since count meta-features of the same order of magnitude carry similar information, we group them so they can

share weights. We do this by bucketing the count meta-features according to their (floored) $\log_2$ value. As this effectively puts the lowest count values, of which there are many, into a different bucket, we optionally introduce a second (ceilinged) bucket to assure smoother transitions. Both buckets are then weighted according to the $\log_2$ fraction lost by the corresponding rounding operation.

In the pseudocode in Figure A.1, meta-features are represented as tuples (hash_value, weight). New meta-features are either added (metafeatures.Add) or first joined with the existing meta-features and only then added (metafeatures.Join).

```
function COMPUTE_METAFEATURES(FeatureTargetPair pair)
    // feature-related meta-features
    metafeatures = {}
    metafeatures.Add(Fingerprint(pair.feature_identity), 1.0)
    metafeatures.Add(Fingerprint(pair.feature_type), 1.0)
    log_count = log(pair.feature_count) / log(2)
    bucket1 = floor(log_count)
    bucket2 = ceil(log_count)
    weight1 = bucket2 - log_count
    weight2 = log_count - bucket1
    metafeatures.Add(Hash(bucket1), weight1)
    metafeatures.Add(Hash(bucket2), weight2)

    // target-related meta-features
    metafeatures.Join(Fingerprint(pair.target_identity), 1.0)

    // feature-target-related meta-features
    log_count = log(pair.feature_target_count) / log(2)
    bucket1 = floor(log_count)
    bucket2 = ceil(log_count)
    weight1 = bucket2 - log_count
    weight2 = log_count - bucket1
    metafeatures.Join(Hash(bucket1), weight1)
    metafeatures.Join(Hash(bucket2), weight2)

    return metafeatures
```

Figure A.1: Meta-feature extraction pseudocode

Appendix B

Multinomial Gradient

In Section 4.2.1, we defined the conditional probability $P(\mathbf{y}|\mathbf{x}) = P(w_k|\Phi(w_1^{k-1}))$ for Sparse Non-negative Matrix language models, according to a multinomial distribution, as follows:

$$P_{\text{multi}}(\mathbf{y}|\mathbf{x}) = \frac{(\mathbf{x}^\mathbf{T}\mathbf{M})_{w_k}}{\sum_{t' \in \mathcal{V}} (\mathbf{x}^\mathbf{T}\mathbf{M})_{t'}}$$

Taking the gradient of the log of this probability w.r.t. the adjustment function $A(f,t)$ gives us:

$$\begin{aligned}
\frac{\partial \log P_{\text{multi}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} &= \left(\frac{\partial \log (\mathbf{x}^\mathbf{T}\mathbf{M})_{w_k}}{\partial M_{ft}} - \frac{\partial \log \sum_{t' \in \mathcal{V}} (\mathbf{x}^\mathbf{T}\mathbf{M})_{t'}}{\partial M_{ft}} \right) \frac{\partial M_{ft}}{\partial A_{ft}} \\
&= \left(\frac{1}{(\mathbf{x}^\mathbf{T}\mathbf{M})_{w_k}} \frac{\partial (\mathbf{x}^\mathbf{T}\mathbf{M})_{w_k}}{\partial M_{ft}} - \frac{1}{\sum_{t' \in \mathcal{V}} (\mathbf{x}^\mathbf{T}\mathbf{M})_{t'}} \frac{\partial \sum_{t' \in \mathcal{V}} (\mathbf{x}^\mathbf{T}\mathbf{M})_{t'}}{\partial M_{ft}} \right) M_{ft} \\
&= \left(\frac{x_f y_t}{\hat{y}_{w_k}} - \frac{x_f}{\sum_{t' \in \mathcal{V}} (\mathbf{x}^\mathbf{T}\mathbf{M})_{t'}} \right) M_{ft} \\
&= x_f M_{ft} \left(\frac{y_t}{\hat{y}_{w_k}} - \frac{1}{\sum_{t' \in \mathcal{V}} \hat{y}_{t'}} \right)
\end{aligned}$$

Appendix C

Poisson Gradient

In Section 4.2.3, we defined the conditional probability $P(\mathbf{y}|\mathbf{x}) = P(w_k|\Phi(w_1^{k-1}))$ for Sparse Non-negative Matrix language models, according to $|\mathcal{V}|$ Poisson distributions, as follows:

$$P_{\text{Pois}}(\mathbf{y}|\mathbf{x}) = \prod_{t' \in \mathcal{V}} \hat{y}_{t'}^{y_{t'}} e^{-\hat{y}_{t'}}$$

Taking the gradient of the log of this probability w.r.t. the adjustment function $A(f,t)$ gives us:

$$\begin{aligned}
\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} &= \left(\frac{\partial \sum_{t' \in \mathcal{V}} y_{t'} \log(\mathbf{x^T M})_{t'}}{\partial M_{ft}} - \frac{\partial \sum_{t' \in \mathcal{V}} (\mathbf{x^T M})_{t'}}{\partial M_{ft}} \right) \frac{\partial M_{ft}}{\partial A(f,t)} \\
&= \left(\frac{1}{(\mathbf{x^T M})_{w_k}} \frac{\partial (\mathbf{x^T M})_{w_k}}{\partial M_{ft}} - x_f \right) M_{ft} \\
&= x_f M_{ft} \left(\frac{y_t}{\hat{y}_{w_k}} - 1 \right)
\end{aligned}$$

Appendix D

Distributing Negative Gradients

Over the entire training set, adding $\frac{C_{f*}}{C_{ft}} M_{ft}$ once on the target t that occurs with feature f amounts to the same as traversing all targets t' that co-occur with f in the training set and adding the term M_{ft} to each:

$$M_{ft} \sum_{(f,t') \in \mathcal{T}} x_f = \frac{C_{f*}}{C_{ft}} M_{ft} C_{ft} = \frac{C_{f*}}{C_{ft}} M_{ft} \sum_{(f,t') \in \mathcal{T}} x_f y_{t'}$$

Applying this to the second term of the Poisson gradient, we get:

$$\begin{aligned} \frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} &= x_f M_{ft} \frac{y_t}{\hat{y}_{w_k}} - x_f M_{ft} = x_f M_{ft} \frac{y_t}{\hat{y}_{w_k}} - x_f y_t M_{ft} \frac{C_{f*}}{C_{ft}} \\ &= x_f y_t M_{ft} \left(\frac{1}{\hat{y}_{w_k}} - \frac{C_{f*}}{C_{ft}} \right) \end{aligned}$$

Appendix E

Leave-one-out Training

In leave-one-out training we exclude the event that generates the gradients from the counts used to compute those gradients. More specifically, for each training example (f, t) we let:

$$C_{f*} \leftarrow C_{f*} - 1 \qquad \text{if } x_f = 1$$

$$C_{ft} \leftarrow C_{ft} - 1 \qquad \text{if } x_f = 1, y_t = 1$$

which means that the gradients for the positive and the negative examples are changed in a different way. Since Eq. (4.14) expresses the general update rule for both type of examples, we first have to separate it into updates for negative and positive examples and then adapt accordingly.

In particular, the second term of Eq. (4.14), i.e. $-x_f y_t M_{ft} \frac{C_{f*}}{C_{ft}}$ is a distribution of $C_{f*} - C_{ft}$ negative and C_{ft} positive updates over C_{ft} positive examples:

$$\begin{aligned} -x_f y_t M_{ft} \frac{C_{f*}}{C_{ft}} &= -x_f y_t M_{ft} \left(\frac{C_{f*} - C_{ft}}{C_{ft}} + \frac{C_{ft}}{C_{ft}} \right) \\ &= -x_f y_t M_{ft} \frac{C_{f*} - C_{ft}}{C_{ft}} - x_f y_t M_{ft} \end{aligned}$$

Furthermore, recall that the first term of Eq. (4.14), i.e. $\frac{x_f y_t M_{ft}}{\hat{y}_{w_k}}$ is non-zero only for positive examples, so it can be added to the positive updates. We can then apply leave-one-out to positive and negative updates separately, ending up with:

$$\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f y_t \left((\frac{1}{\hat{y}^+_{w_k}} - 1) M^+_{ft} - \frac{C_{f*} - C_{ft}}{C_{ft}} M^-_{ft} \right)$$

where M_{ft}^{-}, M_{ft}^{+} and $\hat{y}_{w_k}^{+}$ are defined as follows:

$$M_{ft}^{-} = e^{A(f,t,C_{f*}-1,C_{ft})} \frac{C_{ft}}{C_{f*}-1}$$

$$M_{ft}^{+} = e^{A(f,t,C_{f*}-1,C_{ft}-1)} \frac{C_{ft}-1}{C_{f*}-1}$$

$$\hat{y}_{w_k}^{+} = (\mathbf{x}^T \mathbf{M}^{+})_{w_k}$$

Bibliography

[1] Alder, M., Togneri, R., and Attikiouzel, Y. Dimension of the speech space. *IEE Proc. Communications, Speech and Vision 138*, 3 (1991), 207–214. pages 99

[2] Atal, B. S., and Schroeder, M. R. Predictive coding of speech signals and subjective error criteria. *IEEE Transactions on Acoustics, Speech and Signal Processing 27*, 3 (1978), 247–254. pages 10

[3] Baayen, H., Piepenbrock, R., and Gulikers, L. The CELEX lexical database (release 2) [CD-ROM], Linguistic Data Consortium, Philadelphia, 1995. pages 36, 38

[4] Baum, L. E. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In *Inequalities III: Proceedings of the Third Symposium on Inequalities* (1972), O. Shisha, Ed., Academic Press, pp. 1–8. pages 12

[5] Bellegarda, J. Exploiting latent semantic information in statistical language modeling. *Proc. IEEE 88*, 8 (2000), 1279–1296. pages 5, 45

[6] Bellman, R. *Dynamic Programming*, 1st ed. Princeton University Press, Princeton, NJ, USA, 1957. pages 14

[7] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. A neural probabilistic language model. *Journal of Machine Learning Research 3* (2003), 1137–1155. pages 6, 45

[8] Booij, G. *The morphology of Dutch*. Oxford University Press Inc., Oxford, UK, 2002. pages 28

[9] Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. Large language models in machine translation. In *Proc. EMNLP* (2007), pp. 858–867. pages 76

[10] Brousseau, J., Foster, G., Isabelle, P., Kuhn, R., Normandin, Y., and Plamondon, P. French speech recognition in an automatic dictation system for translators: The TransTalk project. In *Proc. Eurospeech* (1995), pp. 193–196. pages 96

[11] Brown, P. F., Chen, S. F., Della Pietra, S. A., Della Pietra, V. J., Kehler, S., and Mercer, R. L. Automatic speech recognition in machine aided translation. *Computer Speech and Language 8*, 3 (1994), 177–187. pages 96

[12] Brown, P. F., de Souza, P. V., Mercer, R. L., Della Pietra, V. J., and Lai, J. C. Class-based n-gram models of natural language. *Computational Linguistics 18* (1992), 467–479. pages 25, 34

[13] Chelba, C., and Jelinek, F. Structured language modeling. *Computer Speech and Language 14*, 4 (2000), 283–332. pages 6

[14] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One billion word benchmark for measuring progress in statistical language modeling. In *Proc. Interspeech* (2014), pp. 2635–2639. pages xx, 7, 74, 76, 77, 79, 81

[15] Chelba, C., and Pereira, F. Multinomial loss on held-out data for the sparse non-negative matrix language model. *arXiv:1511.01574 [cs.CL]* (2016). pages 80

[16] Chelba, C., and Shazeer, N. Sparse non-negative matrix language modeling for geo-annotated query session data. In *Proc. ASRU* (2015), pp. 8–14. pages 80

[17] Chen, S. F., and Goodman, J. An empirical study of smoothing techniques for language modeling. Tech. Rep. TR-10-98, Computer Science Group, Harvard U., Cambridge, MA, 1998. pages 4, 25, 105

[18] Chen, S. F., Seymore, K., and Rosenfeld, R. Topic adaptation for language modeling using unnormalized exponential models. In *Proc. ICASSP* (1998), pp. 681–684. pages 2

[19] Chen, X., Liu, X., Gales, M., and Woodland, P. Recurrent neural network language model training with noise contrastive estimation for speech recognition. In *Proc. ICASSP* (2015), pp. 5411–5415. pages 73

[20] Chomsky, N. *Syntactic Structures*. Mouton and Co., The Hague, The Netherlands, 1957. pages 1

[21] Clarkson, P. R., and Robinson, A. J. Language model adaptation using mixtures and an exponentially decaying cache. In *Proc. ICASSP* (1997), pp. 799–802. pages 46, 53

[22] Coccaro, N., and Jurafsky, D. Towards better integration of semantic predictors in statistical language modeling. In *Proc. ICSLP* (1998), pp. 2403–2406. pages 48, 51

[23] Davis, S. B., and Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing 28*, 4 (1980), 357–366. pages 10

[24] De Wachter, M., Demuynck, K., Van Compernolle, D., and Wambacq, P. Data driven example based continuous speech recognition. In *Proc. Interspeech* (2003), pp. 1133–1136. pages 10

[25] Deepa, S. R., Bali, K., Ramakrishnan, A. G., and Talukdar, P. P. Automatic generation of compound word lexicon for Hindi speech synthesis. In *Proc. LREC* (2004), pp. 2143–2146. pages 28

[26] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*, 6 (1990), 391–407. pages 45, 46

[27] Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society 39*, 1 (1977), 1–38. pages 12

[28] Demuynck, K. *Extracting, modelling and combining information in speech recognition*. PhD thesis, K.U.Leuven ESAT, 2001. pages 15, 87, 88, 105

[29] Demuynck, K., Laureys, T., and Gillis, S. Automatic generation of phonetic transcriptions for large speech corpora. In *Proc. ICSLP* (2002), vol. 1, pp. 333–336. pages 36, 87, 88, 105

[30] Demuynck, K., Laureys, T., Van Compernolle, D., and Van hamme, H. FLaVoR: A flexible architecture for LVCSR. In *Proc. Eurospeech* (2003), pp. 1973–1976. pages 83, 84

[31] Demuynck, K., Puurula, A., Van Compernolle, D., and Wambacq, P. The ESAT 2008 system for N-Best Dutch speech recognition benchmark. In *Proc. ASRU* (2009), pp. 339–343. pages 36, 86, 87, 105

[32] Demuynck, K., Van Compernolle, D., and Van hamme, H. Robust phone lattice decoding. In *Proc. ICSLP* (2006), pp. 1622–1625. pages 84, 86

[33] Denes, P., and Mathews, M. V. Spoken digit recognition using time-frequency pattern matching. *The Journal of the Acoustical Society of America 32*, 11 (1960), 1450–1455. pages 1

[34] Despres, J., Fousek, P., Gauvain, J.-L., Gay, S., Josse, Y., Lamel, L., and Messaoudi, A. The joint LIMSI and Vecsys research systems for NBEST 2008. In *Proc. N-Best Workshop* (2008). pages 86

[35] Dragsted, B., Mees, I., and Gorm Hansen, I. Speaking your translation : Students' first encounter with speech recognition technology. *Translation & Interpreting 3*, 1 (2011), 10–43. pages 96

[36] Duchateau, J., Demuynck, K., and Van hamme, H. Evaluation of phone lattice based speech decoding. In *Proc. Interspeech* (2009), pp. 1179–1182. pages 84, 90

[37] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12* (2011), 2121–2159. pages 71

[38] Dumais, S. T. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, & Computers 23*, 2 (1991), 229–236. pages 47

[39] Elman, J. L. Finding structure in time. *Cognitive Science 14*, 2 (1990), 179–211. pages 7, 73

[40] Emami, A. *A neural syntactic language model*. PhD thesis, Johns Hopkins University, 2006. pages 6

[41] Ganchev, K., and Dredze, M. Small statistical models by random feature mixing. In *Proc. ACL-2008 Workshop on Mobile Language Processing* (2008), pp. 19–20. pages 69

[42] Good, I. The population frequencies of species and the estimation of population parameters. *Biometrika 40* (1953), 237–264. pages 4

[43] Goodman, J. T. A bit of progress in language modeling, extended version. Tech. Rep. MSR-TR-2001-72, Microsoft Research, 2001. pages 116

[44] Goodman, J. T. Classes for fast maximum entropy training. In *Proc. ICASSP* (2001), pp. 561–564. pages 73, 76

[45] Gutmann, M., and Hyvärinen, A. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research 13*, 1 (2012), 307–361. pages 73

[46] Hermansky, H. Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America 87*, 4 (1990), 1738–1752. pages 10

[47] Hochreiter, S. Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Institut für Informatik, Technische Universität Munich, 1991. pages 7

[48] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780. pages 7

[49] Huang, X., Acero, A., and Hon, H.-W. *Spoken language processing: A guide to theory, algorithm, and system development*, 1st ed. Prentice Hall PTR, New Jersey, USA, 2001. pages 12

[50] Huang, X., Alleva, F., Hwang, M.-Y., and Rosenfeld, R. An overview of the SPHINX-II speech recognition system. *Computer Speech and Language 2* (1993), 137–148. pages 5, 65

[51] Ide, N. Introduction to the special issue on word sense disambiguation: The state of the art. *Computational Linguistics 24* (1998), 1–40. pages 26

[52] Jelinek, F., Bahl, L. R., and Mercer, R. L. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory 21*, 3 (1975), 250–256. pages 2

[53] Jelinek, F., and Mercer, R. L. Interpolated estimation of Markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice* (1980), pp. 381–397. pages 4

[54] Jelinek, F., Mercer, R. L., Bahl, L. R., and Baker, J. K. Perplexity – A measure of the difficulty of speech recognition tasks. *Journal of the Acoustical Society of America 62* (1977), S63. pages 20

[55] Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. Exploring the limits of language modeling. *arXiv:1602.02410 [cs.CL]* (2016). pages 8

[56] Katz, S. M. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions*

on Acoustics, Speech and Signal Processing 35, 3 (1987), 400–401. pages 4, 25, 75

[57] Kessens, J., and van Leeuwen, D. A. N-Best: the Northern- and Southern-Dutch benchmark evaluation of speech recognition technology. In *Proc. Interspeech* (2007), pp. 1354–1357. pages 86

[58] Khadivi, S., and Ney, H. Integration of automatic speech recognition and machine translation in computer-assisted translation. *IEEE Transactions on Audio, Speech and Language Processing 16*, 8 (2008), 1551–1564. pages 96

[59] Khadivi, S., Zolnay, A., and Ney, H. Automatic text dictation in computer-assisted translation. In *Proc. Interspeech* (2005), pp. 2265–2268. pages 96

[60] Klakow, D. Log-linear interpolation of language models. In *Proc. ICSLP* (1998), pp. 1695–1698. pages 64

[61] Kneser, R., and Ney, H. Improved backing-off for m-gram language modeling. In *Proc. ICASSP* (1995), pp. 181–184. pages 25, 75, 105

[62] Koehn, P. Europarl: A parallel corpus for statistical machine translation. In *Proc. Machine Translation Summit* (2005), pp. 79–86. pages 105

[63] Koehn, P., Och, F. J., and Marcu, D. Statistical phrase-based translation. In *Proc. HLT-NAACL* (2003), pp. 48–54. pages 101

[64] Kuhn, R., and de Mori, R. A cache-based natural language model for speech recognition. *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 6 (1990), 570–583. pages 45, 46

[65] Langford, J., Li, L., and Strehl, A. Vowpal Wabbit online learning project (http://hunch.net/?p=309), 2007. pages 69

[66] Laureys, T., Vandeghinste, V., and Duchateau, J. A hybrid approach to compounds in LVCSR. In *Proc. ICSLP* (2002), vol. 1, pp. 697–700. pages 28, 30

[67] Le, H.-S., Oparin, I., Allauzen, A., Gauvain, J.-L., and Yvon, F. Structured output layer neural network language models for speech recognition. *IEEE Transactions on Audio, Speech & Language Processing 21* (2013), 195–204. pages 73

[68] Lee, S.-I., Chatalbashev, V., Vickrey, D., and Koller, D. Learning a meta-level prior for feature relevance from multiple related tasks. In *Proc. ICML* (2000), pp. 489–496. pages 68

[69] Lindgren, N. Machine recognition of human language part I - Automatic speech recognition. *IEEE Spectrum 2*, 3 (1965), 114–136. pages 1

[70] Mikolov, T. *Statistical language models based on neural networks*. PhD thesis, Brno University of Technology, 2012. pages 5, 7, 76

[71] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv:1301.3781 [cs.CL]* (2013). pages 45, 48, 49

[72] Mikolov, T., Deoras, A., Povey, D., Burget, L., and Cernocký, J. Strategies for training large scale neural network language models. In *Proc. ASRU* (2011), pp. 196–201. pages 70, 73

[73] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Proc. NIPS* (2013), pp. 3111–3119. pages 45, 48, 49, 50

[74] Mikolov, T., Yih, W.-t., and Zweig, G. Linguistic regularities in continuous space word representations. In *Proc. HLT-NAACL* (2013), pp. 746–751. pages 45, 48, 49

[75] Mohri, M. Finite-state transducers in language and speech processing. *Computational Linguistics 23*, 2 (1997), 269–311. pages 86

[76] Mohri, M., Pereira, F., and Riley, M. Weighted finite-state transducers in speech recognition. *Computer Speech and Language 16*, 1 (2002), 69–88. pages 15

[77] Morin, F., and Bengio, Y. Hierarchical probabilistic neural network language model. In *Proc. AISTATS* (2005), pp. 246–252. pages 49, 73, 76

[78] Ney, H., Essen, U., and Kneser, R. On structuring probabilistic dependences in stochastic language modeling. *Computer Speech and Language 8* (1994), 1–38. pages 5, 65

[79] Nussbaum-Thom, M., El-Desoky Mousa, A., Schlüter, R., and Ney, H. Compound word recombination for German LVCSR. In *Proc. Interspeech* (2011), pp. 1449–1452. pages 28, 30

[80] Och, F. J., and Ney, H. A systematic comparison of various statistical alignment models. *Computational Linguistics 29*, 1 (2003), 19–51. pages 105

[81] Oostdijk, N. The Spoken Dutch Corpus. Overview and first evaluation. In *Proc. LREC* (2000), pp. 887–894. pages 36, 38, 105

[82] Ordelman, R., van Hessen, A., and de Jong, F. Compound decomposition in Dutch large vocabulary speech recognition. In *Proc. Eurospeech* (2003), pp. 225–228. pages 28, 30

[83] Paulik, M., Fügen, C., Stüker, S., Schultz, T., Schaaf, T., and Waibel, A. Document driven machine translation enhanced ASR. In *Proc. Interspeech* (2005), pp. 2261–2264. pages 96

[84] Pelemans, J., Demuynck, K., Van hamme, H., and Wambacq, P. Coping with language data sparsity: Semantic head mapping of compound words. In *Proc. ICASSP* (2014), pp. 141–145. pages 31, 33, 39

[85] Pelemans, J., Demuynck, K., Van hamme, H., and Wambacq, P. Improving n-gram probabilities by compound-head clustering. In *Proc. ICASSP* (2015), pp. 5221–5225. pages 31

[86] Pelemans, J., Shazeer, N., and Chelba, C. Pruning sparse non-negative matrix n-gram language models. In *Proc. Interspeech* (2015), pp. 1433–1437. pages 80

[87] Pelemans, J., Shazeer, N., and Chelba, C. Sparse non-negative matrix language modeling. *Transactions of the Association for Computational Linguistics 4* (2016), 329–342. pages 80

[88] Pelemans, J., Shazeer, N., and Chelba, C. Sparse non-negative matrix language modeling. In *Proc. EMNLP* (2016). pages 80

[89] Pickhardt, R., Gottron, T., Körner, M., Wagner, P. G., Speicher, T., and Staab, S. A generalized language model as the combination of skipped n-grams and modified Kneser-Ney smoothing. In *Proc. ACL* (2014), pp. 1145–1154. pages 65

[90] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. The Kaldi speech recognition toolkit. In *Proc. IEEE 2011 Workshop on Automatic Speech Recognition and Understanding* (2011). pages 15

[91] Reddy, A. M., and Rose, R. C. Integration of statistical models for dictation of document translations in a machine-aided human translation task. *IEEE Transactions on Audio, Speech and Language Processing 18*, 8 (2010), 2015–2027. pages 96

[92] Reddy, D. R., Erman, L. D., Fennell, R. D., and Neely, R. B. The Hearsay speech understanding system. In *Proc. 3rd Int. Joint Conf. Artificial Intelligence* (1973), pp. 185–193. pages 1

[93] Řehůřek, R., and Sojka, P. Software framework for topic modelling with large corpora. In *Proc. LREC* (2010), pp. 45–50. pages 52

[94] Réveil, B., and Martens, J.-P. Reducing speech recognition time and memory use by means of compound (de-) composition. In *Proc. ProRISC* (2008), pp. 348–352. pages 28, 30

[95] Rodríguez, L., Reddy, A. M., and Rose, R. C. Efficient integration of translation and speech models in dictation based machine aided human translation. In *Proc. ICASSP* (2012), pp. 4949–4952. pages 96, 97, 100, 108

[96] Rosenfeld, R. *Adaptive statistical language modeling: A maximum entropy approach*. PhD thesis, Carnegie Mellon University, 1994. pages 4, 5, 45, 65

[97] Rosenfeld, R., Chen, S. F., and Zhu, X. Whole-sentence exponential language models: A vehicle for linguistic-statistical integration. *Computer Speech and Language 15* (2001), 55–73. pages 2

[98] Sainath, T. N., Weiss, R. J., Senior, A. W., Wilson, K. W., and Vinyals, O. Learning the speech front-end with raw waveform CLDNNs. In *Proc. Interspeech* (2015), pp. 1–5. pages 9

[99] Salton, G. *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971. pages 47

[100] Schwenk, H. Continuous space language models. *Computer Speech and Language 21* (2007), 492–518. pages 6

[101] Schwenk, H., and Gauvain, J.-L. Neural network language models for conversational speech recognition. In *Proc. ICSLP* (2004), pp. 1215–1218. pages 73

[102] Schwenk, H., and Gauvain, J.-L. Training neural network language models on very large corpora. In *Proc. EMNLP* (2005), pp. 201–208. pages 73

[103] Shazeer, N., Pelemans, J., and Chelba, C. Sparse non-negative matrix language modeling for skip-grams. In *Proc. Interspeech* (2015), pp. 1428–1432. pages 80

[104] Singh, M., and Klakow, D. Comparing RNNs and log-linear interpolation of improved skip-model on four Babel languages: Cantonese, Pashto, Tagalog, Turkish. In *Proc. ICASSP* (2013), pp. 8416–8420. pages 65

[105] Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. Semantic compositionality through recursive matrix-vector spaces. In *Proc. EMNLP* (2012), pp. 1201–1211. pages 56

[106] Steinbiss, V., Tran, B.-H., and Ney, H. Improvements in beam search. In *Proc. ICSLP* (1994), pp. 2143–2146. pages 15, 88

[107] Stolcke, A. SRILM – An extensible language modeling toolkit. In *Proc. ICSLP* (2002), pp. 257–286. pages 52, 105

[108] Tan, M., Zhou, W., Zheng, L., and Wang, S. A scalable distributed syntactic, semantic, and lexical language model. *Computational Linguistics 38*, 3 (2012), 631–671. pages 5, 78, 79

[109] van den Bosch, A., Busser, B., Canisius, S., and Daelemans, W. An efficient memory-based morphosyntactic tagger and parser for Dutch. In *Computational Linguistics in the Netherlands 2006: Selected papers from the seventeenth CLIN Meeting* (Utrecht, The Netherlands, 2007), P. Dirix, Ed., LOT, pp. 191–206. pages 38

[110] Vandeghinste, V. Lexicon optimization: Maximizing lexical coverage in speech recognition through automated compounding. In *Proc. LREC* (2002), pp. 1270–1276. pages 34, 40

[111] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory 13*, 2 (1967), 260–269. pages 14

[112] Williams, W., Prasad, N., Mrva, D., Ash, T., and Robinson, T. Scaling recurrent neural network language models. In *Proc. ICASSP* (2015), pp. 5391–5395. pages 7, 18, 73, 74

[113] Witten, I. H., and Bell, T. C. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory 37*, 4 (1991), 1085–1094. pages 25

[114] Woods, W. A., Makhoul, J., Wolf, J., and Rovner, P. Organizing a system for continuous speech understanding. In *85th Meeting of the Acoustical Society of America* (1973). pages 1

[115] Xu, P., Gunawardana, A., and Khudanpur, S. Efficient subsampling for training complex language models. In *Proc. EMNLP* (2011), pp. 1128–1136. pages 70, 73

[116] Yu, D., Deng, L., and Acero, A. The maximum entropy model with continuous features. In *Proc. NIPS Workshop* (2008). pages 5

[117] Zhou, J., Shi, Q., and Qin, Y. Generating compound words with high order n-gram information in large vocabulary speech recognition systems. In *Proc. ICASSP* (2011), pp. 5560–5563. pages 28

Short Biography

Joris Pelemans was born on August 10, 1980 in Mechelen, Belgium. He received a B.Sc. degree in Computer Science in 2000, a M.Sc. degree in Computer Science in 2002 and a M.Sc. degree in Artificial Intelligence (option Speech and Language Technology) in 2003 from KU Leuven, Belgium. He then worked in IT for several years before joining the ESAT-PSI Speech group, KU Leuven in October 2010, where he received a Ph.D. degree in Engineering Science in May, 2017. He's the author and co-author of 4 journal papers and more than 15 conference papers. His research interests are in natural language processing, with a focus on language modeling, automatic speech recognition and machine translation.

List of Publications

Articles in International Journals

1. **Joris Pelemans**, Kris Demuynck, Hugo Van hamme and Patrick Wambacq, "*The Effect of Word Similarity on N-gram Language Models in Northern and Southern Dutch*", CLIN Journal, volume 4, pages 91–104, December 2014.

2. Yangyang Shi, Martha Larson, **Joris Pelemans**, Catholijn M. Joncker, Patrick Wambacq, Pascal Wiggers and Kris Demuynck, "*Integrating Meta-Information into Recurrent Neural Network Language Models*", Speech Communication, volume 73, pages 64–80, September 2015.

3. Lyan Verwimp, **Joris Pelemans**, Hugo Van hamme and Patrick Wambacq, "*Expanding N-gram Training Data for Language Models based on Morpho-syntactic Transformations*", CLIN Journal, volume 5, pages 49–64, November 2015.

4. **Joris Pelemans**, Noam Shazeer and Ciprian Chelba, "*Sparse Non-negative Matrix Language Modeling*", Transactions of the Association for Computational Linguistics, volume 4, pages 329–342, July 2016.

Articles in International Conferences

1. **Joris Pelemans**, Kris Demuynck and Patrick Wambacq, "*A Layered Approach for Dutch Large Vocabulary Continuous Speech Recognition*", In Proc. ICASSP, pages 4421–4424, Kyoto, Japan, March 2012.

2. **Joris Pelemans**, Kris Demuynck and Patrick Wambacq, "*Dutch Automatic Speech Recognition on the Web: Towards a General Purpose*

System", In Proc. Interspeech, pages 2123–2126, Portland, USA, September 2012 (Best Show & Tell Presentation Award).

3. Emre Yılmaz, **Joris Pelemans**, Stefan Lievens and Hugo Van hamme, "*Speech Reception Threshold Measurement Using Automatic Speech Recognition*", In Proc. ICASSP, Florence, Italy, May 2014.

4. **Joris Pelemans**, Kris Demuynck, Hugo Van hamme and Patrick Wambacq, "*Speech Recognition Web Services for Dutch*", In Proc. LREC, pages 3041–3044, Reykjavik, Iceland, May 2014.

5. **Joris Pelemans**, Kris Demuynck, Hugo Van hamme and Patrick Wambacq, "*Coping with Language Data Sparsity: Semantic Head Mapping for Compound Words*", In Proc. ICASSP, pages 141–145, Firenze, Italy, May 2014.

6. Emre Yılmaz, **Joris Pelemans** and Hugo Van hamme, "*Automatic Assessment of Children's Reading with the FLaVoR Decoding Using a Phone Confusion Model*", In Proc. Interspeech, pages 969–972, Singapore, September 2014.

7. **Joris Pelemans**, Kris Demuynck, Hugo Van hamme and Patrick Wambacq, "*Improving N-gram Probabilities by Compound-head Clustering*", In Proc. ICASSP, pages 5221–5225, Brisbane, Australia, April 2015.

8. Noam Shazeer, **Joris Pelemans** and Ciprian Chelba, "*Sparse Non-negative Matrix Language Modeling For Skip-grams*", In Proc. Interspeech, pages 1428–1432, Dresden, Germany, September 2015.

9. **Joris Pelemans**, Noam Shazeer and Ciprian Chelba, "*Pruning Sparse Non-negative Matrix N-gram Language Models*", In Proc. Interspeech, pages 1433–1437, Dresden, Germany, September 2015.

10. **Joris Pelemans**, Tom Vanallemeersch, Kris Demuynck, Hugo Van hamme and Patrick Wambacq, "*Efficient Language Model Adaptation for Automatic Speech Recognition of Spoken Translations*", In Proc. Interspeech, pages 2262–2266, Dresden, Germany, September 2015.

11. **Joris Pelemans**, Tom Vanallemeersch, Lyan Verwimp, Kris Demuynck, Hugo Van hamme and Patrick Wambacq, "*Language Model Adaptation for ASR of Spoken Translations using Phrase-based Translation Models and Named Entity Models*", In Proc. ICASSP, pages 5985–5989, Shanghai, China, March 2016.

12. **Joris Pelemans**, Lyan Verwimp, Kris Demuynck, Hugo Van hamme and Patrick Wambacq, "*SCALE: A Scalable Language Engineering Toolkit*", In Proc. LREC, pages 3868–3871, Portorož, Slovenia, May 2016.

13. Lyan Verwimp, Brecht Desplanques, Kris Demuynck, **Joris Pelemans**, Marieke Lycke and Patrick Wambacq, "*STON: Efficient Subtitling in Dutch Using State-of-the-Art Tools*", In Proc. Interspeech, pages 780–781, San Francisco, USA, September 2016.

14. Sofoklis Kakouros, **Joris Pelemans**, Lyan Verwimp, Patrick Wambacq and Okko Räsänen, "*Analyzing the Contribution of Top-down Lexical and Bottom-up Acoustic Cues in the Detection of Sentence Prominence*", In Proc. Interspeech, pages 1074–1078, San Francisco, USA, September 2016.

15. **Joris Pelemans**, Noam Shazeer and Ciprian Chelba, "*Sparse Non-negative Matrix Language Modeling*", In Proc. EMNLP, Austin, Texas, USA, November 2016.

16. Lyan Verwimp, **Joris Pelemans**, Hugo Van hamme and Patrick Wambacq, "*Character-Word LSTM Language Models*", In Proc. EACL, pages 417–427, Valencia, Spain, April 2017.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING (ESAT)
CENTER FOR PROCESSING SPEECH AND IMAGES (PSI)
Kasteelpark Arenberg 10
B-3001 Heverlee
joris.pelemans@esat.kuleuven.be
http://www.esat.kuleuven.be